#14 UT 10/9/03

PATENT
DECLARATION UNDER
37 CFR §1.131
EXAMINING GROUP NO.
2645

IN THE UNITED STATES PATENT & TRADEMARK OFFICE

Applicants:	Pirasteh et al.	: Paper No:	
Serial No.	09/223,993	: Group Art Unit:	2645
Filed:	December 31, 1998	: Examiner:	Gerald Gauthier

For: SYSTEM AND APPARATUS FOR IVR PORT SHARING

Mail Stop AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. 1.131

Dear Sir:

Hassan Pirasteh and Jeffrey J. Sager, the inventors in the above-identified patent application (hereinafter, the Inventors), declare as follows:

1. The Bjornberg patent, U.S. 6,366,658, was granted (and became available to the public) on Apr. 2, 2002, and was filed on May 7, 1998. A portion of this patent describes a telecommunications architecture for call center services using advanced interactive voice response service nodes that pass call destination information out of band to an interactive voice recognition (IVR) application.
2. The Inventors filed a patent application covering their invention, as claimed in the above-identified patent application, on June 8, 1998.
3. Prior to May 7, 1998, the Inventors conceived of and reduced their invention to practice.
4. Prior to May 7, 1998, the Inventors documented a description of a "Port Sharing" infrastructure using a basic CTI components eventually described in the Patent Application. A copy of this documented description is attached to this Declaration. The attached pages are unaltered from their original form except that all dates have been masked. (2003 §1.131 Declaration - Exhibit A).
5. Prior to May 7, 1998, the Inventors were present at a meeting wherein a reduction to practice occurred of the precise integration point between the IVR and the CTI system where ANI, DNIS, and other potential data could be forwarded to the IVR before the voice connection

was routed by the PBX switch to the chosen IVR port. (2003 §1.131 Declaration – Exhibit A).

6. Prior to May 7, 1998, the Inventors caused source code to be written incorporating the inventive interface described in the patent application. (2003 §1.131 Declaration – Exhibit A).
7. Prior to May 7, 1998, the Inventors commissioned the drafting of a patent application covering their invention from the law firm of Standley and Gilchrist.
8. Prior to May 7, 1998, the Inventors both prepared a draft specification and revised the draft specification for the benefit of the patent attorney.
9. This draft specification includes draft claims that were eventually modified into the claims filed in the Inventors' patent application on June 8, 1998.
10. The declarants state that all statements made herein of actual knowledge are true, or if made on information and belief are believed to be true.
11. The declarants further state that all statements made herein were made with knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that any such willful false statements may jeopardize the validity of this application or any patent issuing thereon.

Hassan Pirasteh Date: 9/15/03

Jeffrey J. Sager Date: 9/15/03

CinLibrary/1319252.1

IVR Port Sharing with Intuity Conversant and Nabnasset VESP at Matrixx Marketing Inc.

Hassan Pirasteh
J.J. Sager

[redacted]

TAB 1: Overview

IVR Port Sharing

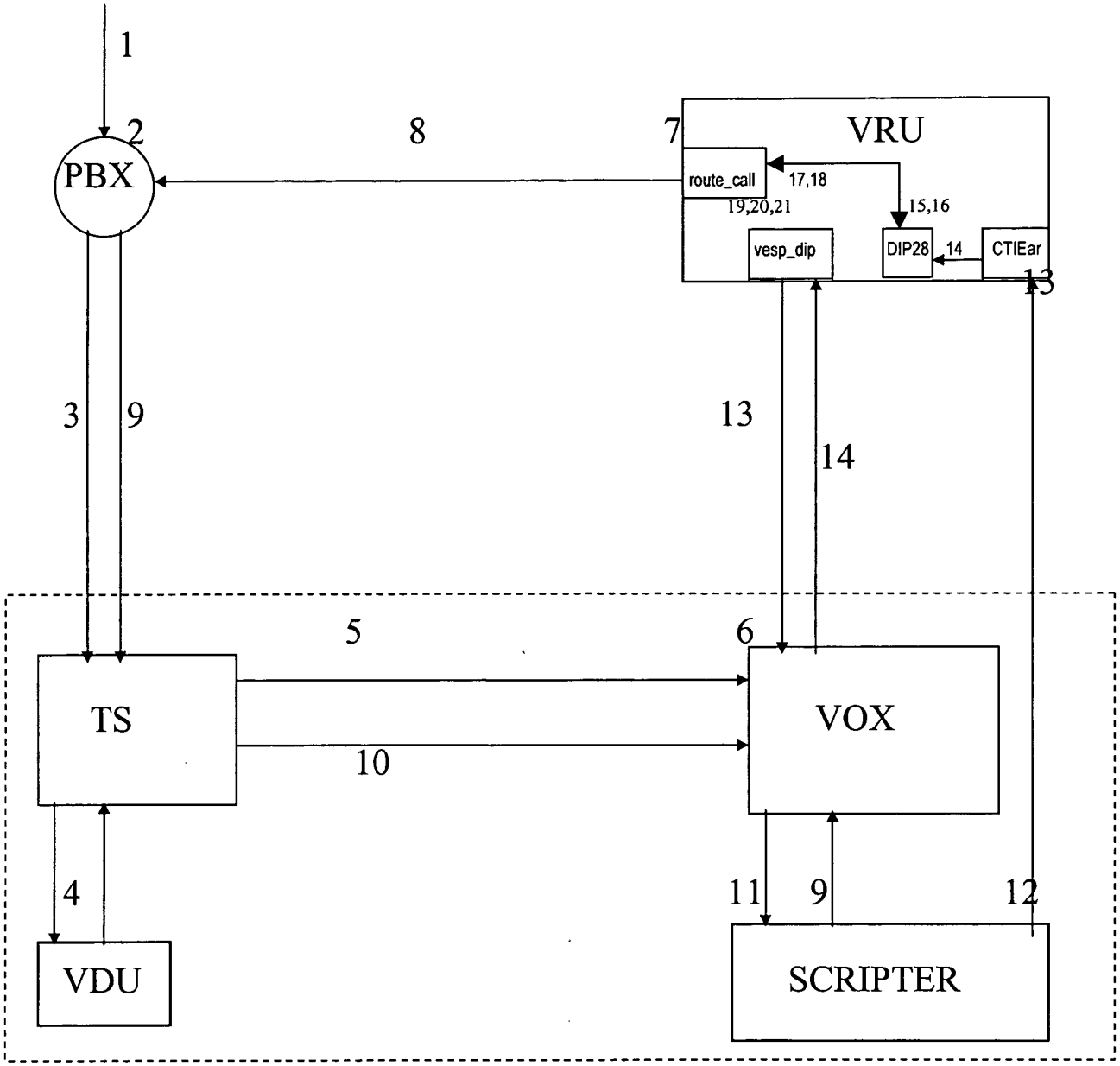
In order to make the most efficient use of Conversant® ports, we have developed a method of dynamic port allocation based on DNIS (i.e. the incoming 800 number). In a normal installation like ours where the IVR voice channels are wired to the station side of the PBX switch, each IVR channel would have a single application program hard-assigned. Then these channels would be grouped into hunt-groups (from the PBX perspective) so that calls for each application would be routed to the hunt group for a specific client application. The channels in that hunt group could answer calls only for that client application and no other. This type of arrangement, while functional, is very inefficient because the number of channels assigned to a client's hunt group has to be large enough to handle the peak call volumes - during lower call volume periods, many of the channels go unused. A more efficient method of IVR channel use is to group all channels of an IVR box into one hunt group and then assign the client application dynamically, as the call arrives. This way, the channels are busy handling the mixed call volumes of many client projects and are available for any peak period of any project. We developed our own unique "port sharing" method in order to make maximum use of a limited IVR port resource.

In order to do dynamic port allocation, the IVR must be made aware of the incoming number that was called (usually an 800 number) before the call is actually answered. Our "port sharing" method has the DNIS information passed to the IVR out-of-band (on a separate digital link) so that the IVR can allocate the correct application in a very short period of time (typically less than 500 milliseconds after the call arrives on the PBX switch). The basic data flow for port sharing is this:

1. **The Call arrives on the PBX.**
2. **The PBX passes the DNIS and ANI information on to the telephony server.**
3. **The server formats the information and sends it to a background process on the IVR.**
4. **The background process determines that the message is meant for a port on this particular box and saves the DNIS and ANI data in memory.**
5. **The call arrives at the IVR port.**
6. **A special application that is always hard-assigned to every IVR port notices the port has an arriving call and asks the background process for the DNIS and ANI information.**
7. **The special application looks the DNIS up in a table, determines the correct application and executes that application on the port.**

Redacted

Diagram of Port Sharing Solution using Nabnasset VESP Services



Redacted

Detailed Functional Description

1. A call arrives at the PBX switch.
2. The switch follows the vector instructions and queues the caller to a Conversant ACD split.
3. The PBX notifies the VESP TS server of a new call which is being directed to a monitored extension.
4. The TS server requests that the VDU server create a new VDU and assign a unique VDU ID.
5. The TS Server notified the VOX server that a call destined for a VRU port has arrived (the VOX is responsible for maintaining a state model of all VRU ports).
6. The VOX server waits to see that the call arrives at the VRU port.
7. The voice port of the Conversant box recognizes an incoming call either via ring current (analog ports) or T1 messaging (Line Side T1 ports). The Conversant automatically starts the application program assigned to every port called route_call.
8. The route_call application takes the phone off-hook.
9. The PBX informs the TS Server that the extension has gone off-hook.
10. The TS Server informs the VOX server of the off-hook state change.
11. The VOX Server issues a Script.Qualify1 method call to the Scripter Server. (This is translated internally to a scripter_custom_qualify1() function call. The scripter_custom_qualify1() function is avaliable for development by the CTI development team in the custom.c source.)
12. The Scripter maintains an internal table of all of the Conversant machines monitored by the VOX server. It creates an ISDN-formatted message that includes the ANI, DNIS, CALLED_EXTENSION, and AGENT_EXTENSION information and passes this message on to all Conversants using a Remote Procedure Call (RPC).
13. The remote procedure (to_d28()) is contained in a program running on the Conversant machines called CTIEar. The RPC facility is maintained in such a way that the TCP channel opened by the RPC is left open indefinitely to speed subsequent message passing.
14. The CTIEar program receives the message and passes it onto a UNIX message queue that is monitored by a program called dip28_isdn.
15. The dip28_isdn program receives the message from the message queue and determines if it is meant for this particular Conversant box by matching the AGENT_EXTENSION against a list of the extensions known to be wired to this Conversant.
16. If the message was in fact meant for this Conversant box, the data is stored in memory for later retrieval.
17. The route_call application requests the ISDN information from dip28_isdn using the get_ani External Function call.
18. The dip28_isdn program returns the information to route_call. If the information has not yet arrived (a rare occurrence) the route_call application sleeps for a small time period and asks for the information again and eventually the information is passed.
19. The route_call application parses the information and uses the DNIS value to perform an Oracle table lookup.
20. Assuming the Oracle table (ROUTE_APPL) contains an entry for this DNIS, a field in the table contains the name of the application program associated with that DNIS and route_call executes that application.
21. If any of this protocol breaks down, the route_call application is programmed to ask the caller to re-enter the 800 number they have dialed and it uses that response to search the ROUTE_APPL table via a different field.

Redacted

TAB 2: Source Code

File custom.c - User-customizable section of Nabnasset Scripter Server

Line 124: A memory table is created to hold a list of all Conversants known to VESP along with their RPC handle.

Line 263 and Line 1238: Upon Initialization, a list of Conversants is loaded into the memory table.

Line 557: The scripter_custom_qualify1 service is called automatically for every telephone call destined for a Conversant port. The ANI and DNIS information is formatted and sent to all known Conversants using RPC.

File CTIEar.x - RPC definition for "to_d28()" remote procedure

File _CTIEar.c - Remote procedure implementation

The CTIEar procedure runs on the Conversant boxes. It is called remotely from the VESP Scripter and receives the ISDN-formatted message. It passes this message on to the dip28_isdn process via message queues.

File isdn.h - Defines for dip28_isdn.c

File dip28_isdn.c - Conversant DIP process

This DIP is responsible for parsing the ISDN-formatted message that was passed on via CTIEar. It determines if the message was meant for the Conversant box it is running on by comparing the extension of the port that is receiving the call with a list of the extensions wired to this Conversant box. If the message is meant for this box, the data is stored in memory for retrieval by the application programs.

File route_call - ScriptBuilder application program assigned to all shared Conversant ports

The route_call program is assigned to all shared Conversant ports. When a call arrives, route_call asks dip28_isdn for the ISDN information. The DNIS is parsed and a table lookup is made. If the DNIS is found in the table, the appropriate user application is started. If the DNIS is not found, the caller is asked to enter the 800 number they have just dialed and a second search is done.

File get_ani.t - External Function used by route_call

Redacted

```
/*****************************************************************************
*  Copyright (c) [redacted] AT&T Solutions Customer Care
*                All rights Reserved
*****************************************************************************
*
* File: custom.c
* Type: ASAI
*       MLINK
*
* Description:
* This custom.c file implements the VESP Scripter for the ATTS-CC Call Centers.
* It borrows heavily from the Nabnasset example Scripter for a fictional
* call center at the Fourth National Bank of Acton.  This version of custom.c
* will operate with both the ASAI Telephony Server for Lucent Definity
* switch, and the Meridian Link Telephony Server for Nortel.  One single
* definition differs for the two switches, and is determined by a single
* #ifdef at the top of this file.
*
* Revision History:
* Date      Who      Rev    Comments
* ========================================================================
* [redacted]  JMeyers  1.0.0  Created
* [redacted]  JSager   1.0.1  Made into ATTS-CC version.  I'm of a mind right
*                       now to leave in all of Jim's Bank example for
*                       future reference.
* [redacted]  JSager   1.0.2  Port Sharing: Utilize Scripter_custom_qualify1 to
*                       alert Conversants of incoming call.  Take out
*                       trans2 routines that attempted to control VRU
*                       ports - Nabnasset won't let us control ports
*                       for which TS doesn't know split number.
* [redacted]  JSager   1.0.3  Tidy up the incoming-call section to handle
*                       route requests that really aren't requesting
*                       a route-to destination.
* [redacted]  JSager   1.1.0  Add FedEx route request handling and tr2 trans
*                        for Vantive Interface.
* [redacted]  JSager   1.1.1  Redefine Dnis Table parsing for our
*                        multi-customer/multi-call-center environment.
*                        The _dnisEnglish table now has the following
*                        format:
*                  Client:CallCenter:PrimaryRoute:SecondaryRoute:Description
*
*****************************************************************************/
#include <stdio.h>
#include <time.h>
#include <ctype.h>

#include <orb.h>
#include "script.h"

/* Added for RPC to Conversant */
typedef u_long u_int32;
#include <rpc/rpc.h>
#include "cvis_intfc/CTIEar.h"

/* The Meridian Link TS does not send "dnis" as a couple with the incoming call
 * event unless the CDN that generated the route request was a DNIS CDN.  For
 * this example, we assume a different CDN is used for every call purpose, so
 * the CDN is the number we want to look up in the "DNIS" table.  The CDN
 * always appears as "dest" in the incoming call event.
 */

```

Redacted

```
#ifdef MERIDIAN
#define DNIS_COUPLE_NAME "dest"
#else
#define DNIS_COUPLE_NAME "dnis"
#endif

/*
 * Global variables.
 */

char Empty_String[] = ""; /* Default for extensions */
char *version = "ATTS-CC Scripter 1.1.1";

/* "From" enumeration for how a call entered the Scripter */
typedef enum {
    FR_UNKNOWN = 0,
    FR_INCOMING,
    FR_TRANSACTION
    } FromEnum;

/* Enumeration for Clients */
typedef enum {
    UNKNOWN = 0,
    FEDEXN,
    GE_TPC
    } ClientEnum;

/* Matching text strings to store in VDU */
char *ClientText[] = {
    "Unknown",
    "FedEx NEXUS",
    "GE TPC"
    };

/* This structure holds data associated with a particular DNIS */
typedef struct Dnis_Data {
    char dnis[33];          /* DNIS */
    ClientEnum client;  /* Which client owns the call */
    char cc_location[4]; /* Call Center Location designation */
    char prim_dnis_route[6];  /* Primary routing */
    char sec_dnis_route[6];  /* Secondary routing */
    char dnis_desc[65];   /* Description of this DNIS (64 chars) */
    } Dnis_Data;
Dnis_Data *dnis_table = NULL;
int num_dnis_table = 0;   /* Number of entries in dnis_tbl */

/*
 * Instance data for Vesp_Request() calls.
 */
typedef struct Callback_Data {
    char vdu_id[33];
    char ani[11];
    FromEnum from;          /* How did call get here? INCOMING or TRANSACTION? */
    ClientEnum client;  /* Which client owns call */
    char prim_route[6];        /* Primary Routing extension */
    char sec_route[6];        /* Secondary Routing extension */
    SeqCouple *vdu_data;  /* Data eventually bound for VDU.SetValues */
    Request request;      /* Handle for sending deferred requests */
    SeqCouple *tr2_result;/* Handle to output argument of tr2 transaction */
    char dnis_desc[65];
    } Callback_Data;
```

Redacted

```
123
124 /*
125  * List of Conversant names.  Used by qualify1 routine to pass ANI and DNIS
126  * to all Conversants in this list.  This list is filled in at start-up.
127  */
128 typedef struct Cvis_List {
129       char  cvis[64];
130       CLIENT      *handle;
131 } Cvis_List;
132 Cvis_List *cvis_table = NULL;
133 int NumCvisList = 0;
134
135 /* Prototypes for custom-defined transaction functions */
136 void tr2_MeterLookUp (Object object, Environment *ev, VDU_ID vduid,
137                           _IDL_SEQUENCE_string *arglist,
138                           _IDL_SEQUENCE_Couple *resultlist);
139 void tr2_AccountLookUp (Object object, Environment *ev, VDU_ID vduid,
140                           _IDL_SEQUENCE_string *arglist,
141                           _IDL_SEQUENCE_Couple *resultlist);
142 void tr2_ContactLookUp (Object object, Environment *ev, VDU_ID vduid,
143                           _IDL_SEQUENCE_string *arglist,
144                           _IDL_SEQUENCE_Couple *resultlist);
145
146 /* Prototypes for functions that respond to CORBA events */
147 ULONG Scripter_custom_incomingcall_event (Scripter_Data *scripter_data,
148                                   SeqCouple *data);
149 ULONG Scripter_custom_disconnect_event (Scripter_Data *scripter_data,
150                                   SeqCouple *data);
151 ULONG Scripter_custom_server_term_event (Scripter_Data *scripter_data,
152                                   SeqCouple *eventinfo);
153 ULONG Scripter_custom_other_event (Scripter_Data *scripter_data, char *name,
154                                   SeqCouple *data);
155
156 /* Prototypes for functions that respond to other things */
157 void Scripter_custom_update (void);
158 ORBStatus  Scripter_custom_poll (void);
159 void Scripter_custom_exit (void);
160 void Scripter_custom_exception (void);
161 void Scripter_custom_qualify1 (Object object,
162                                   Environment *ev,
163                                   VDU_ID vduid,
164                                   _IDL_SEQUENCE_Couple *arglist,
165                                   _IDL_SEQUENCE_Couple *resultlist);
166
167 /* Prototypes for callback functions */
168 ULONG cb_VDUSetV_Route_Prim (Identifier interface, ORBStatus status,
169             Environment *ev, long user_data, Session session,
170             char *vdu_id, SeqCouple *data);
171
172 ULONG cb_VDUSetV_ANILookUp (Identifier interface, ORBStatus status,
173             Environment *ev, long user_data, Session session,
174             char *vdu_id, SeqCouple *data);
175
176 ULONG TS_Route_callback( Identifier interface, ORBStatus status,
177             Environment *ev, long user_data, Session session,
178             char *vdu_id, char *ext);
179
180 ULONG cb_TSRoute_rmCB( Identifier interface, ORBStatus status,
181             Environment *ev, long user_data, Session session,
182             char *vdu_id, char *ext);
183
```

Redacted

```
184 ULONG VDU_Terminate_callback( Identifier interface, ORBStatus status,
185             Environment *ev, long user_data, Session session,
186             char *vdu_id);
187
188 ULONG FEDEX_LookUp_callback(Identifier interface,
189             ORBStatus status, Environment *ev, long user_data,
190             Session session, char *vdu_id, char *data);
191
192 ULONG FEDEX_ANILookUp_callback(Identifier interface,
193             ORBStatus status, Environment *ev, long user_data,
194             Session session, char *vdu_id, char *data);
195
196 /* Prototypes for miscellaneous functions */
197 void parse_dnis_data (Dnis_Data **dnis_table, int *num_dnis_table);
198 void parse_cvis_data (Cvis_List **cvis_table, int *numcvislist);
199 ORBStatus parse_config_parameters (void);
200 Couple *find_couple (SeqCouple *seqcpl, char *name);
201
202
203
204 /*****************************************************************************
205  Function: Scripter_custom_init
206
207  Description:
208  This function runs automatically when the scripter is initialized.  It
209  performs function registrations and initialization tasks.
210
211  Arguments:
212  Object object: Object pointer for this server
213
214  Returns:
215  VESP_SUCCESS
216 *****************************************************************************/
217 extern ULONG Scripter_custom_init (Object object)
218 {
219     /* This is where we register function names to be called when
220      * various things occur:
221      */
222
223     /* Functions to respond to events from the TS */
224     Script_register_custom_incomingcall (Scripter_custom_incomingcall_event);
225     Script_register_custom_disconnect (Scripter_custom_disconnect_event);
226     Script_register_custom_server_term (Scripter_custom_server_term_event);
227
228     /* Function to respond to other events */
229     Script_register_custom_other (Scripter_custom_other_event);
230
231     /* Function to execute when Script.GenericUpdate method is invoked
232      * ("Update Server" button clicked in ManCon)
233      */
234     Script_register_custom_update (Scripter_custom_update);
235
236     /* Function to call once each time through the VESP poll loop */
237     Script_register_custom_poll (Scripter_custom_poll);
238
239     /* Functions to call when exiting or crashing */
240     Script_register_custom_exit (Scripter_custom_exit);
241     Script_register_custom_exception (Scripter_custom_exception);
242
243     /* Register a Qualify1 function */
244     Script_register_custom_qualify1 (Scripter_custom_qualify1);
```

Redacted

```
245
246     /* Register the users custom transaction functions */
247     Script_register_trans2_routine ("MeterLookUp", tr2_MeterLookUp);
248     Script_register_trans2_routine ("AccountLookUp", tr2_AccountLookUp);
249     Script_register_trans2_routine ("ContactLookUp", tr2_ContactLookUp);
250
251     /* One-time initialization tasks to perform when the scripter
252      * comes up:
253      */
254
255     /* This version number overwrites the "base" Scripter version */
256     vesp_set_server_version (version);
257
258     /* Download the DNIS table from the DS and parse it into an
259      * array of structures.
260      */
261     parse_dnis_data (&dnis_table, &num_dnis_table);
262
263     /*
264      * Download list of Conversants known to this VESP environment
265      */
266     parse_cvis_data (&cvis_table, &NumCvisList);
267
268     parse_config_parameters ();
269
270     return (VESP_SUCCESS);
271 }
272
273
274 /********************** Event Handling Functions ***************************/
275
276 /*****************************************************************************
277  Function: Scripter_custom_incomingcall_event
278
279  Description:
280  This function runs automatically when the scripter receives an incoming call
281  event from the TS (TS.IncomingCall.event).
282
283  Arguments:
284  Scripter_Data *scripter_data: This structure is passed to maintain source
285     code compatibility for all switches.  The only useful data it contains
286     in the ASAI and Meridian Link Scripters is a valid session to use in
287     calls to Vesp_Request().
288
289  Note: In the case of the ASAI and Meridian Link Scripters, this structure
290     contains NO PERSISTENT DATA, even though the structure definition in
291     script.h implies that it might.  This structure is persistent through the
292     life of the call ONLY for the Aspect version of the Scripter; no such
293     persistence exists for the ASAI and Meridian Link versions!  When
294     Scripter_custom_incomingcall_event() is through, the structure is gone.
295     Do not pass pointers to it to Vesp_Request callback functions!
296
297  SeqCouple *eventinfo: Sequence of couples passed by the TS with the event.
298     Contains members such as ani, dnis, dest, orig, etc.
299
300  Returns:
301  VESP_SUCCESS
302 *****************************************************************************/
303 ULONG Scripter_custom_incomingcall_event (
304                              Scripter_Data *scripter_data,
305                              SeqCouple *eventinfo)
```

Redacted

```
306 {
307     Callback_Data *callback_data;
308     char *ani, *dnis, *vdu_id, *cc_loc;
309     int i;
310
311     /* Allocate a callback structure.  This structure will continue through
312      * all asynchronous calls (Vesp_Request) until the call is finally
313      * routed.  It will be freed at that time.
314      */
315     callback_data = vesp_calloc (1, sizeof (Callback_Data));
316
317     /* Now create a sequence of couples to start loading VDU data into.
318      * Eventually we will call VDU.SetValues, but we keep a running
319      * list until that time.  The list is part of the callback_data
320      * structure in case we have to call some other method meanwhile.
321      */
322     callback_data->vdu_data = vesp_couple_seq_create();
323
324     /* Extract useful members from the event for easy access */
325     vesp_dup_seq_couple_value (eventinfo, "ani", &ani);
326     vesp_dup_seq_couple_value (eventinfo, DNIS_COUPLE_NAME, &dnis);
327     vesp_dup_seq_couple_value (eventinfo, "vdu_id", &vdu_id);
328
329     /* Fill the callback structure with any data we know so far */
330     strcpy (callback_data->vdu_id, vdu_id);
331     callback_data->from = FR_INCOMING;  /* Tag the starting point */
332
333     /* Initialize the dnis-related data with valid defaults */
334     callback_data->client = UNKNOWN;
335     strcpy (callback_data->prim_route, Empty_String);
336     strcpy (callback_data->sec_route, Empty_String);
337     strcpy (callback_data->dnis_desc, "Non-Adjunct-Routed DNIS");
338
339     /* Search the dnis table for this call's dnis.  Overwrite the
340      * default data in the callback structure if it is found.
341      */
342     for (i = 0; i < num_dnis_table; i++)
343         {
344         if (!strcmp (dnis, dnis_table[i].dnis))
345             {
346             callback_data->client = dnis_table[i].client;
347             strcpy (callback_data->prim_route, dnis_table[i].prim_dnis_route);
348             strcpy (callback_data->sec_route, dnis_table[i].sec_dnis_route);
349             strcpy (callback_data->dnis_desc, dnis_table[i].dnis_desc);
350           cc_loc = dnis_table[i].cc_location;
351             break;
352             }
353         }
354
355     /* Add a few items to the vdu data for writing the VDU later */
356     vesp_couple_seq_add_couple_values (callback_data->vdu_data,
357        "Client", ClientText[callback_data->client]);
358     vesp_couple_seq_add_couple_values (callback_data->vdu_data,
359         "dnis_desc", callback_data->dnis_desc);
360     vesp_couple_seq_add_couple_values (callback_data->vdu_data,
361         "call_ctr", cc_loc);
362
363     /* Branch here based on Client who owns this DNIS */
364     switch (callback_data->client)
365     {
366
```

Redacted

```
367         /* Client is unknown */
368         case UNKNOWN:
369
370           /*
371            * We probably got here because DNIS was not in the dnis_table.
372            * This is not necessarily an error, it could be that they
373            * just wanted collected digits to show up in the VDU.  We will
374            * populate the VDU with default data and tell the switch to
375            * route to the default VDN (Should be NULL which means to try the
376            * next vector step)
377            */
378
379            /* Write data to the VDU.  The specified callback function
380             * will route the call afterwards.
381             */
382            Vesp_Request ("VDU.SetValues", cb_VDUSetV_Route_Prim,
383                (ULONG)callback_data, scripter_data->session,
384                callback_data->vdu_id, callback_data->vdu_data);
385            break;
386
387         /* Client is GE TPC */
388         case GE_TPC:
389           /*
390            * We don't actually do any adjunct routing for GE TPC currently,
391            * so we'll just update the VDU and route the call to the
392            * Default VDN (NULL - which will cause the vector to move on
393            * to the next step).
394            */
395              Vesp_Request ("VDU.SetValues", cb_VDUSetV_Route_Prim,
396                     (ULONG)callback_data, scripter_data->session,
397                     callback_data->vdu_id, callback_data->vdu_data);
398
399              break;
400
401         /* Client is FedEx NEXUS */
402         case FEDEXN:
403           /* Set some default values in the VDU then do an ANI lookup */
404           /*
405            * NOTE: [redacted] - for now we are not doing the ANI lookup.
406            * Some code has been commented out below for this change.  See
407            * further comments below.
408            */
409           strncpy(callback_data->ani, ani, 10);
410           vesp_couple_seq_add_couple_values (callback_data->vdu_data,
411             "UANI", ani);
412           vesp_couple_seq_add_couple_values (callback_data->vdu_data,
413             "UAPP", "VANTIVE");
414           vesp_couple_seq_add_couple_values (callback_data->vdu_data,
415             "CALLORIGIN", "NEXUS");
416
417       /*
418        * NOTE CONTINUED: [redacted] -
419        * When we're ready to do the ANI lookup, uncomment the next two
420        * vesp_couple_seq_add_couple_values and the Vesp_Request with a
421        * callback to cb_VDUSetV_ANILookUp and then delete the
422        * Vesp_Request with the callback to cb_VDUSetV_Route_Prim.
423        */
424           /* vesp_couple_seq_add_couple_values (callback_data->vdu_data, */
425       /*    "OBJECTNAME", Empty_String); */
426          /* vesp_couple_seq_add_couple_values (callback_data->vdu_data, */
427       /*    "OBJECTNUM", Empty_String); */
```

Redacted

```
428
429          /* Vesp_Request("VDU.SetValues", cb_VDUSetV_ANILookUp, */
430       /*     (ULONG)callback_data, scripter_data->session, */
431       /*     callback_data->vdu_id, callback_data->vdu_data); */
432
433              Vesp_Request ("VDU.SetValues", cb_VDUSetV_Route_Prim,
434                     (ULONG)callback_data, scripter_data->session,
435                     callback_data->vdu_id, callback_data->vdu_data);
436
437              break;
438
439         default:
440              break;
441
442     } /* End switch (callback_data->dnis_data.client) */
443
444     /* Free memory allocated by vesp_dup_seq_couple_value() */
445     vesp_free (ani);
446     vesp_free (dnis);
447     vesp_free (vdu_id);
448
449     return (VESP_SUCCESS);
450 }
451
452
453 /* Call has disconnected.  This event will be delivered by the Aspect
454  * CallCenter version of the VESP Telephony Server, but not by the
455  * Lucent Definity or Nortel Meridian Link versions.
456  */
457 ULONG Scripter_custom_disconnect_event (Scripter_Data * scripter_data,
458                                SeqCouple *eventinfo)
459 {
460     return (VESP_SUCCESS);
461 }
462
463
464 /* The TS has failed or exited.  The Scripter will automatically try to
465  * reassign to the TS (repeatedly), and generally that will bring the TS
466  * back up.  Any additional custom code for this event would go here.
467  */
468 extern ULONG Scripter_custom_server_term_event (Scripter_Data *scripter_data,
469                                SeqCouple *eventinfo)
470 {
471     log_unparsable ("Dectected TS failure!!!");
472     return (VESP_SUCCESS);
473 }
474
475
476 /* Some "other" event has been received.  If the Scripter assigns to some
477  * custom server to receive events, this function will be executed when
478  * events are received.  TS.SendData events (Aspect switch only) also
479  * cause this function to be called.  Parse the "name" argument to
480  * determine what event occurred.
481  */
482 ULONG Scripter_custom_other_event (Scripter_Data *scripter_data, char *name,
483                                SeqCouple *eventinfo)
484 {
485     log_unparsable ("Unexpected event received: %s", name);
486     return (VESP_SUCCESS);
487 }
488
```

Redacted

```
489
490 /* The GenericUpdate method has been invoked.  This generally means an
491  * administrator selected the Scripter from ManCon and clicked on the
492  * "Update Server" button.
493  */
494 void Scripter_custom_update ()
495 {
496     /* Reload the DNIS table from the DS and parse it into an
497      * array of structures.  Free the old one first.
498      */
499     vesp_free (dnis_table);
500     parse_dnis_data (&dnis_table, &num_dnis_table);
501
502     /*
503      * Download list of Conversants known to this VESP environment
504      */
505     vesp_free (cvis_table);
506     parse_cvis_data (&cvis_table, &NumCvisList);
507
508     log_unparsable ("Scripter has been updated");
509 }
510
511
512 /* This function is called automatically once every time through the main
513  * VESP polling loop.  Put any periodic functionality here.
514  */
515 #define TIMECHECK_INTERVAL  (5*60) /* Every five minutes */
516
517 ULONG Scripter_custom_poll ()
518 {
519     static time_t last_time_check = 0;    /* Last time */
520     time_t now;
521
522     /* Check the local time.  If TIMECHECK_INTERVAL has passed, log
523      * a quick message to the log file.
524      */
525     now = time (NULL);
526     if (now > last_time_check + TIMECHECK_INTERVAL) {
527       vesp_log_info(VESP_CAT_ALL, "HEARTBEAT", "TEST");
528       log_unparsable("Heap Size=%ld",vesp_get_heap_size());
529       vesp_heap_check();
530         last_time_check = now;
531         }
532
533     return (VESP_SUCCESS);
534 }
535
536
537 /* The Scripter is exiting normally.  This generally means an administrator
538  * selected the Scripter from ManCon and clicked on the "Stop Server"
539  * button.  Put any cleanup code here.
540  */
541 void Scripter_custom_exit ()
542 {
543     log_unparsable ("Scripter is exiting normally");
544 }
545
546
547 /* The Scripter is exiting abnormally.  This could indicate a segment
548  * violation, a "kill" executed from the command line, or other system
549  * problem.  Put any cleanup code here.
```

Redacted

```
 */
void Scripter_custom_exception ()
{
    log_unparsable ("Scripter is exiting with an exception");
}


/* The method Script.Qualify1 has been invoked, presumably by the VOX server.
 * When the switch sends calls straight to the VRU without a VESP route
 * request (SOP at ATTS-CC), we can start call qualification in this function.
 * We do Conversant port-sharing here by sending dnis and ani data to the
 * monitored Conversants via an RPC pipe.
 */
void Scripter_custom_qualify1 (Object object,
                               Environment *ev,
                               VDU_ID vduid,
                               _IDL_SEQUENCE_Couple *arglist,
                               _IDL_SEQUENCE_Couple *resultlist)
{
    char *ani, *dnis, *dest, *called, *orig;
    char *loc_ani, *loc_dnis; /* Might need to change values of dnis and ani.*/
                        /* Using non-vesp variables allows us to still */
                        /* free the original memory space allocated by */
                        /* the toolkit.                                */
    char    *cvis;
    CLIENT  *cl;
    int           *rc;
    char    *str_ptr, isdn_string[41];
    int           ani_type, ani_len, cnt;

    /* Collect data that would be useful for Conversant Port Sharing */
    vesp_dup_seq_couple_value(arglist, "ani", &ani);
    vesp_dup_seq_couple_value(arglist, "dnis", &dnis);
    vesp_dup_seq_couple_value(arglist, "dest", &dest);
    vesp_dup_seq_couple_value(arglist, "called", &called);
    vesp_dup_seq_couple_value(arglist, "orig", &orig);

    /*
     * Filter that data.  Sometimes, when called from within PBX we get
     * no dnis so we'll use the originating number instead.  Also, we
     * don't get inforation about the "ani type" (inside extension, trunk
     * group, or real ANI) so we just try to figure it out as best we can.
     */
    if(strlen(dnis) == 0)
      loc_dnis=called;
    else
      loc_dnis=dnis;
    ani_len=strlen(ani);
    if(ani_len == 0) {
      ani_type=1;
      loc_ani=orig;
    }
    else if(ani_len == 5 || ani_len == 4) {
      ani_type=1;
      loc_ani=ani;
    }
    else if(ani_len == 3 || ani_len == 2 || ani_len == 1) {
      ani_type=2;
      loc_ani=ani;
    }
    else {
```

Redacted

```
611       ani_type=9;
612       loc_ani=ani;
613     }
614
615     /* Set up data string to look like original IG gateway string */
616     sprintf(isdn_string,"A0000000001%5s%1d%10s1%5s1%5s", dest, ani_type,
617             loc_ani, called, loc_dnis);
618     str_ptr=isdn_string;
619
620     /*
621      * Find names and RPC handles of Conversants to send to in Cvis_List.
622      */
623     for(cnt=0 ; cnt < NumCvisList ; cnt++)
624     {
625       cvis=cvis_table[cnt].cvis;
626       cl=cvis_table[cnt].handle;
627
628       vesp_log_info(VESP_CAT_ALL, "VESP_TO_CVIS", cvis);
629       log_unparsable("ISDN String='%s'", isdn_string);
630
631       if(cl == NULL) {
632             /*
633              * We have not opened a channel to this cvis yet.
634              */
635               vesp_log_info(VESP_CAT_ALL, "OPEN_RPC_CHANNEL", cvis);
636             cl=clnt_create(cvis, CTIEARPROG, CTIEARVERS, "tcp");
637             if(cl == NULL) {
638                       vesp_log_info(VESP_CAT_ALL,"RPC_CREATE_ERROR",cvis);
639                       log_unparsable(clnt_spcreateerror(cvis));
640                       log_unparsable("CTIEar may not be running on %s", cvis);
641                       continue;
642             }
643             cvis_table[cnt].handle=cl; /* Remember this handle */
644       }
645
646       /* Call the remote procedure that passes string on to DIP28 */
647       rc=to_d28_1(&str_ptr, cl);
648       if(rc == NULL || *rc != 0) {
649                   /************************************
650                    * There is room for improvement here.
651                    * If we discover that the once-open
652                    * channel to this cvis is now closed,
653                    * we currently mark it that way in
654                    * our table, but we don't try to
655                    * re-open it and send again.  This
656                    * results in a lost message.  We will
657                    * open a new one with the "next"
658                    * message.
659                    ************************************/
660             clnt_destroy(cl);
661             vesp_log_info(VESP_CAT_ALL,"RPC_WRITE_ERROR",cvis);
662             log_unparsable("Destroy old channel to %s",cvis);
663             cvis_table[cnt].handle=NULL;
664       }
665
666     } /* for all entries in the cvis_list table */
667
668     vesp_free(ani);
669     vesp_free(dnis);
670     vesp_free(dest);
671     vesp_free(called);
```

Redacted

```
672     vesp_free(orig);
673 }
674
675
676 /******************** Custom transaction functions ************************/
677
678 /*
679  * One of the FedEx-NEXUS/Vantive interface transactions.  This transaction
680  * is used for Meter number lookups in the Vantive databse.  The request is
681  * forwarded to the FEDEX Server for processing.
682  */
683 void tr2_MeterLookUp (Object object, Environment *ev, VDU_ID vduid,
684                         _IDL_SEQUENCE_string *arglist,
685                         _IDL_SEQUENCE_Couple *resultlist)
686 {
687     Environment  local_ev;
688     Callback_Data *callback_data;
689     char    meter_num[41];
690
691     /* Get Meter Number from Conversant */
692     strncpy(meter_num,arglist->_buffer[0],sizeof(meter_num)-1);
693
694     /* If Conversant failed to send us a meter number, let's just short
695      * circuit it right here.
696      */
697     if(strlen(meter_num) < 1)
698     {
699       vesp_log_info(VESP_CAT_ALL, "Error", "Bad Input to tr2_MeterLookUp");
700       return;
701     }
702
703     callback_data = vesp_calloc (1, sizeof (Callback_Data));
704
705     callback_data->request = Object_duplicate (ev->request, &local_ev);
706     Request_defer (ev->request, ev);
707
708     /*
709      * In case we have a result to send back later...
710      */
711     callback_data->tr2_result = resultlist;
712
713     Vesp_Request ("FEDEX.MeterLookUp", FEDEX_LookUp_callback,
714             (ULONG)callback_data, ev->session,
715             vduid, meter_num);
716
717     return;
718 }
719
720 /*
721  * One of the FedEx-NEXUS/Vantive interface transactions.  This transaction
722  * is for Account number lookups in the Vantive database.  The request is
723  * forwarded to the FEDEX Server for processing.
724  */
725 void tr2_AccountLookUp (Object object, Environment *ev, VDU_ID vduid,
726                         _IDL_SEQUENCE_string *arglist,
727                         _IDL_SEQUENCE_Couple *resultlist)
728 {
729     Environment  local_ev;
730     Callback_Data *callback_data;
731     char    acct_num[20];
732
```

Redacted

```
733     /* Get Account Number from Conversant */
734     strncpy(acct_num,arglist->_buffer[0],sizeof(acct_num)-1);
735
736     /* If Conversant failed to send us a meter number, let's just short
737      * circuit it right here.
738      */
739     if(strlen(acct_num) < 1)
740     {
741       vesp_log_info(VESP_CAT_ALL, "Error", "Bad Input to tr2_AccountLookUp");
742       return;
743     }
744
745     callback_data = vesp_calloc (1, sizeof (Callback_Data));
746
747     callback_data->request = Object_duplicate (ev->request, &local_ev);
748     Request_defer (ev->request, ev);
749
750     /*
751      * In case we have a result to send back later...
752      */
753     callback_data->tr2_result = resultlist;
754
755     Vesp_Request ("FEDEX.AccountLookUp", FEDEX_LookUp_callback,
756             (ULONG)callback_data, ev->session,
757             vduid, acct_num);
758
759     return;
760 }
761
762 /*
763  * One of the FedEx-NEXUS/Vantive interface transactions.  This transaction
764  * is for Contact number lookups in the Vantive database.  The request is
765  * forwarded to the FEDEX Server for processing.
766  */
767 void tr2_ContactLookUp (Object object, Environment *ev, VDU_ID vduid,
768                         _IDL_SEQUENCE_string *arglist,
769                         _IDL_SEQUENCE_Couple *resultlist)
770 {
771     Environment  local_ev;
772     Callback_Data *callback_data;
773     char    cont_num[21];
774
775     /* Get Account Number from Conversant */
776     strncpy(cont_num,arglist->_buffer[0],sizeof(cont_num)-1);
777
778     /* If Conversant failed to send us a meter number, let's just short
779      * circuit it right here.
780      */
781     if(strlen(cont_num) < 1)
782     {
783       vesp_log_info(VESP_CAT_ALL, "Error", "Bad Input to tr2_ContactLookUp");
784       return;
785     }
786
787     callback_data = vesp_calloc (1, sizeof (Callback_Data));
788
789     callback_data->request = Object_duplicate (ev->request, &local_ev);
790     Request_defer (ev->request, ev);
791
792     /*
793      * In case we have a result to send back later...
```

Redacted

```
794      */
795     callback_data->tr2_result = resultlist;
796
797     Vesp_Request ("FEDEX.ContactLookUp", FEDEX_LookUp_callback,
798             (ULONG)callback_data, ev->session,
799             vduid, cont_num);
800
801     return;
802 }
803
804
805 /********** Callback functions ***********/
806
807 /* This callback from VDU.SetValues routes the call to the primary route dest.*/
808 ULONG cb_VDUSetV_Route_Prim (Identifier interface, ORBStatus status,
809              Environment *ev, long user_data, Session session,
810              char *vdu_id, SeqCouple *data)
811 {
812     Callback_Data *callback_data;
813
814     if (status != VESP_SUCCESS || ev->_major != NO_EXCEPTION)
815         {
816         vesp_log_info(VESP_CAT_ALL, "Error", "VDU.SetValues failure");
817         log_unparsable ("Error was \"%s\"",
818             (ev->vesp_error_description) ?
819             ev->vesp_error_description :
820             vesp_find_error (status));
821
822         /* Some problem occurred while Writing to the VDU.  Don't panic,
823          * just continue and try to route the call anyway.
824          */
825         }
826
827     /* Resurrect the callback structure from user_data */
828     callback_data = (Callback_Data *)user_data;
829
830     /* Use the routing extension in the callback structure to route
831      * the call.
832      */
833     Vesp_Request( "TS.Route", TS_Route_callback,
834           (ULONG)0UL, ev->session,
835           vdu_id, callback_data->prim_route );
836
837     /* The Scripter's involvement in this call is now done.  Since we
838      * touched the VDU for the call earlier, the Scripter is on the list of
839      * "owners" for that VDU.  We invoke the terminate method now.
840      */
841     Vesp_Request( "VDU.Terminate", VDU_Terminate_callback,
842           (ULONG)0UL, ev->session, vdu_id);
843
844     /* The callback data is not needed in TS_Route_callback(), since
845      * there is no more work to do there, so we free it now.  First
846      * free the sequence of VDU couples, then the callback structure.
847      */
848     vesp_couple_seq_delete (callback_data->vdu_data);
849     vesp_free (callback_data);
850
851     return (VESP_SUCCESS);
852 }
853
854 /*
```

Redacted

```
855  * Special FEDEX NEXUS callback from VDU SetValues.  Now we need to do an
856  * ANI lookup in the FEDEX Server.
857  */
858 ULONG cb_VDUSetV_ANILookUp (Identifier interface, ORBStatus status,
859              Environment *ev, long user_data, Session session,
860              char *vdu_id, SeqCouple *data)
861 {
862     Callback_Data *callback_data;
863
864     if (status != VESP_SUCCESS || ev->_major != NO_EXCEPTION)
865         {
866         vesp_log_info(VESP_CAT_ALL, "Error", "VDU.SetValues failure");
867         log_unparsable ("Error was \"%s\"",
868             (ev->vesp_error_description) ?
869             ev->vesp_error_description :
870             vesp_find_error (status));
871
872         /* Some problem occurred while Writing to the VDU.  Don't panic,
873          * just continue and try to perform an ANI Lookup.
874          */
875         }
876
877     /* Resurrect the callback structure from user_data */
878     callback_data = (Callback_Data *)user_data;
879
880     /*
881      * Check the ANI that came with the call.  If it is the system default,
882      * which means the caller had no ANI delivered, route the call to the
883      * secondary routing VDN (Conversant).  Otherwise, submit an ANI Lookup
884      * request to the FEDEX server and make a routing decision based on that
885      * return.  One or the other of the callback functions must remove the
886      * callback structure from memory (this necessitates a special route
887      * callback function - if we tried to remove it from memory here, the
888      * cases where ANILookUp is done would crash the server).
889      */
890     if((strlen(callback_data->ani) < 10)
891       || (!strncmp(callback_data->ani+3,"5556666",7)))
892     {
893       Vesp_Request( "TS.Route", cb_TSRoute_rmCB,
894           (ULONG)callback_data, ev->session,
895           vdu_id, callback_data->sec_route );
896     }
897     else
898     {
899       Vesp_Request ("FEDEX.ANILookUp", FEDEX_ANILookUp_callback,
900             (ULONG)callback_data, ev->session,
901             vdu_id, callback_data->ani);
902     }
903
904     Vesp_Request( "VDU.Terminate", VDU_Terminate_callback,
905           (ULONG)0UL, ev->session, vdu_id);
906
907     return (VESP_SUCCESS);
908 }
909
910 ULONG TS_Route_callback( Identifier interface, ORBStatus status,
911                      Environment *ev, long user_data, Session session,
912                      char *vdu_id, char *ext)
913 {
914     return (VESP_SUCCESS);
915 }
```

Redacted

```
916
917 /*
918  * Special case of Route callback in which we remove the callback structure
919  * from memory.
920  */
921 ULONG cb_TSRoute_rmCB( Identifier interface, ORBStatus status,
922                        Environment *ev, long user_data, Session session,
923                        char *vdu_id, char *ext)
924 {
925     Callback_Data *callback_data;
926
927     callback_data = (Callback_Data *)user_data;
928
929     vesp_couple_seq_delete (callback_data->vdu_data);
930     vesp_free (callback_data);
931
932     return (VESP_SUCCESS);
933 }
934
935 ULONG VDU_Terminate_callback( Identifier interface, ORBStatus status,
936              Environment *ev, long user_data, Session session,
937              char *vdu_id)
938 {
939     return VESP_SUCCESS;
940 }
941
942 ULONG FEDEX_LookUp_callback(Identifier interface,
943             ORBStatus status, Environment *ev, long user_data,
944             Session session, char *vdu_id, char *data)
945 {
946     Callback_Data *callback_data;
947     Environment local_ev;
948     any     result;
949
950     if(status != VESP_SUCCESS || ev->_major != NO_EXCEPTION)
951     {
952       vesp_log_info(VESP_CAT_ALL, "ERROR", "FEDEX LookUp Failed");
953       log_unparsable("Error was \"%s\"",
954                   (ev->vesp_error_description) ?
955                   ev->vesp_error_description :
956                   vesp_find_error(status));
957     }
958
959     /* Resurrect the callback structure from user_data */
960     callback_data = (Callback_Data *)user_data;
961
962     /*
963      * At least for now, the VRU is not expecting any data returned
964      * from the tr2_*LookUp call.
965      */
966     result._type = "void";
967     result._value = (void *)NULL;
968     if (Request_send_deferred_response (callback_data->request,
969         &local_ev, &result) != VESP_SUCCESS)
970     {
971         Vesp_Send_Alarm( object_gbl, "Script.Problem",
972             "high",  "Could not send deferred response" );
973         log_unparsable("Could not send def response from MeterLookUp callback");
974     }
975
976     Object_release( callback_data->request, &local_ev );
```

Redacted

```
 977     vesp_free (callback_data);
 978
 979     return VESP_SUCCESS;
 980 }
 981
 982 ULONG FEDEX_ANILookUp_callback(Identifier interface,
 983             ORBStatus status, Environment *ev, long user_data,
 984             Session session, char *vdu_id, char *ani)
 985 {
 986     Callback_Data *callback_data;
 987
 988     callback_data = (Callback_Data *)user_data;
 989
 990     if(status != VESP_SUCCESS )
 991     {
 992       /*
 993        * The ANI lookup failed or there was no unique match in the
 994        * Vantive DB.  Route the call to the VRU.
 995        */
 996       vesp_log_info(VESP_CAT_ALL, "ANI_NOTUNIQUE", ani);
 997
 998       Vesp_Request( "TS.Route", TS_Route_callback,
 999           (ULONG)0UL, ev->session,
1000           vdu_id, callback_data->sec_route );
1001     }
1002     else
1003     {
1004       /*
1005        * The ANI lookup was unique in the Vantive DB.  Route the call to
1006        * the agent split.
1007        */
1008       vesp_log_info(VESP_CAT_ALL, "ANI_UNIQUE", ani);
1009
1010       Vesp_Request( "TS.Route", TS_Route_callback,
1011           (ULONG)0UL, ev->session,
1012           vdu_id, callback_data->prim_route );
1013     }
1014
1015     /* We're done with the callback structure */
1016     vesp_couple_seq_delete (callback_data->vdu_data);
1017     vesp_free (callback_data);
1018
1019     return (VESP_SUCCESS);
1020 }
1021
1022 /********** Utility functions ***********/
1023
1024 /* Case insensitive version of strncmp() */
1025 int strncmp__i (char *buff1, char *buff2, int len)
1026 {
1027     int i, c1, c2;
1028     for ( i = 0; i < len; ++i, ++buff1, ++buff2)
1029     {
1030     c1 = toupper (*buff1); c2 = toupper (*buff2);
1031     if (c1 == '\0' && c2 == '\0')
1032         break;
1033     if (c1 == c2) continue;
1034     return (c1 > c2) ? 1 : -1;
1035     }
1036     return 0;
1037 }
```

Redacted

```
1038
1039 /* Case insensitive version of strcmp() */
1040 int strcmp__i (char *buff1, char *buff2)
1041 {
1042     int c1, c2;
1043     for (; TRUE; ++buff1, ++buff2)
1044     {
1045     c1 = toupper (*buff1); c2 = toupper (*buff2);
1046     if (c1 == '\0' && c2 == '\0')
1047         break;
1048     if (c1 == c2) continue;
1049     return (c1 > c2) ? 1 : -1;
1050     }
1051     return 0;
1052 }
1053
1054
1055 /* This utility function locates a couple by name in a sequence of
1056  * couples.  A pointer to the matching couple is returned.  No data is
1057  * copied and no memory is allocated.  Beware the returned pointer is
1058  * not used beyond the scope of the SeqCouple it points into!
1059  */
1060 Couple *find_couple (SeqCouple *seqcpl, char *name)
1061 {
1062     int i;
1063
1064     if (seqcpl == NULL || name == NULL)
1065         return NULL;
1066
1067     for (i = 0; i < seqcpl->_length; ++i )
1068         if (!strcmp (seqcpl->_buffer[i].name, name))
1069             return (&seqcpl->_buffer[i]);
1070
1071     return NULL;
1072 }
1073
1074 /* Case-insensitive version of find_couple() using strcmp__i() */
1075 Couple *find_couple_i (SeqCouple *seqcpl, char *name)
1076 {
1077     int i;
1078     if (seqcpl == NULL || name == NULL)
1079     return NULL;
1080
1081     for (i = 0; i < seqcpl->_length; ++i )
1082     if (!strcmp__i (seqcpl->_buffer[i].name, name))
1083         return (&seqcpl->_buffer[i]);
1084
1085     return NULL;
1086 }
1087
1088
1089 /* Parse dnis file and put info into an array of structures */
1090 void parse_dnis_data (Dnis_Data **dnis_tbl, int *num_dnis_tbl)
1091 {
1092     int i, j;
1093     char *client, *cc_loc, *proute, *sroute, *desc, *ptr;
1094     SeqSeqCouple seqseqcp;
1095     Request request;
1096     Environment ev;
1097     Session session;
1098     ULONG result;
```

Redacted

```
1099    Couple *Cptr;
1100
1101    /* Set defaults in case something goes wrong */
1102    *dnis_tbl = NULL;
1103    *num_dnis_tbl = 0;
1104
1105    /* Get the default session for GetFewRecords call */
1106    if (Vesp_get_default_session(object_gbl, &session) != VESP_SUCCESS)
1107        {
1108        log_unparsable ("No default session for loading DNIS table\n");
1109        return;
1110        }
1111
1112    /* Build a request to load DNIS table from the Directory Server.  This
1113     * function is called only at startup or when the GenericUpdate method
1114     * is invoked, so we can safely make a synchronous call to the DS.
1115     */
1116    seqseqcp._maximum = 1; /* Specify maximum of one row to be returned */
1117    seqseqcp._length = 0;
1118    seqseqcp._buffer = NULL;
1119    result = Vesp_Request_Sync("DS.GetFewRecords", &ev, session, &request,
1120        "",                     /* Default table */
1121        "type=_dnisEnglish",    /* Criteria string */
1122        &seqseqcp);             /* Resulting tables */
1123
1124    /* If the DS returned a positive response, process the results */
1125    if (result == VESP_SUCCESS || result == VESP_PARTIAL_SUCCESS)
1126    {
1127        /* Make sure that a row of data was actually returned */
1128        if (seqseqcp._length > 0)
1129        {
1130            /* The number of defined DNIS is the length of the first row
1131             * returned.  Use this to allocate memory for the table of
1132             * structures in memory.  It may be more than we actually use
1133             * if we encounter invalid entries, but no matter.
1134             */
1135            *dnis_tbl = (Dnis_Data *)vesp_calloc (
1136                seqseqcp._buffer[0]._length,
1137                sizeof (Dnis_Data));
1138
1139            /* Now parse each entry in this sequence.  i counts through
1140             * the couples in the sequence, j counts through the structures
1141             * we are creating.
1142             */
1143            for (i = j = 0; i < seqseqcp._buffer[0]._length; i++)
1144            {
1145                /* Assign a pointer to save some typing */
1146                Cptr = &(seqseqcp._buffer[0]._buffer[i]);
1147
1148                /* If name or value is blank, skip this one */
1149                if (strlen (Cptr->name) == 0 || strlen (Cptr->value) == 0)
1150                    continue;
1151
1152                /* Parse the value for
1153             * Client:CallCenter:PrimaryRoute:SecondaryRoute:Description.
1154             * Do not remove any white space, parse based on the colons
1155             * only.  If the format is not right, skip this DNIS.
1156             * Replace the colons with null characters and save pointers
1157             * to the start of each segment of text.
1158                 */
1159
```

```
1160                /* Set a pointer to the start of the Client, then find
1161                 * the colon marking the end.
1162                 */
1163                client = Cptr->value;
1164                for (ptr = client; *ptr && *ptr != ':'; ++ptr)
1165                    ;
1166                if (*ptr == '\0') /* 1st colon missing. Skip this DNIS */
1167                    continue;
1168                *ptr = '\0'; /* Terminate client, replacing the colon */
1169
1170                /* Set a pointer to the start of the Call Center name, then
1171             * find the colon marking the end.
1172                 */
1173                cc_loc = ptr + 1;
1174                for (ptr = cc_loc; *ptr && *ptr != ':'; ++ptr)
1175                    ;
1176                if (*ptr == '\0') /* 2nd colon missing. Skip this DNIS */
1177                    continue;
1178                *ptr = '\0'; /* Terminate proute, replacing the colon */
1179
1180                /* Set a pointer to the start of the primary route, then find
1181                 * the colon marking the end.
1182                 */
1183                proute = ptr + 1;
1184                for (ptr = proute; *ptr && *ptr != ':'; ++ptr)
1185                    ;
1186                if (*ptr == '\0') /* 2nd colon missing. Skip this DNIS */
1187                    continue;
1188                *ptr = '\0'; /* Terminate proute, replacing the colon */
1189
1190                /* Set a pointer to the start of the secondary route, then find
1191                 * the colon marking the end.
1192                 */
1193                sroute = ptr + 1;
1194                for (ptr = sroute; *ptr && *ptr != ':'; ++ptr)
1195                    ;
1196                if (*ptr == '\0') /* 2nd colon missing. Skip this DNIS */
1197                    continue;
1198                *ptr = '\0'; /* Terminate sroute, replacing the colon */
1199
1200                /* Set a pointer to the start of the dnis description */
1201                desc = ptr + 1;
1202
1203                /* Copy the data to the structure.  Convert the client
1204                 * code from the table into an enumerated value.  Copy the
1205                 * dnis, location, routes, and description verbatim.
1206                 */
1207
1208                /* The couple's name is the DNIS */
1209                strcpy ((*dnis_tbl)[j].dnis, Cptr->name);
1210
1211                if (!strncmp__i (client, "FEDEX NEXUS", 11))
1212                    (*dnis_tbl)[j].client = FEDEXN;
1213                else if (!strncmp__i (client, "GE TPC", 6))
1214                    (*dnis_tbl)[j].client = GE_TPC;
1215            else
1216                (*dnis_tbl)[j].client = UNKNOWN;
1217                strcpy ((*dnis_tbl)[j].cc_location, cc_loc);
1218                strcpy ((*dnis_tbl)[j].prim_dnis_route, proute);
1219                strcpy ((*dnis_tbl)[j].sec_dnis_route, sroute);
1220                strcpy ((*dnis_tbl)[j].dnis_desc, desc);
```

Redacted

```
1221
1222                log_unparsable("DNIS data retreived for %s:%s",
1223                  (*dnis_tbl)[j].dnis, (*dnis_tbl)[j].dnis_desc);
1224
1225                ++j;  /* Valid structure filled in, increment count */
1226            } /* End for (i = j = 0...) */
1227
1228            *num_dnis_tbl = j; /* Record the number of valid DNIS's */
1229        } /* End if (seqseqcp._length > 0) */
1230    } /* End if (result == VESP_SUCCESS...) */
1231
1232    /* Now free all resources that were allocated automatically by
1233     * Vesp_Request_Sync().
1234     */
1235    Vesp_Request_Delete(session, request);
1236 }
1237
1238 /*
1239  * parse_cvis_data - creates a list of Conversants known to this VESP
1240  * environment.  It does this by looking at the configuration information
1241  * for VOX and looks for "vru" records.
1242  */
1243 void parse_cvis_data (Cvis_List **cvis_tbl, int *numcvislist)
1244 {
1245     SeqSeqCouple seqseqcp;
1246     Request request;
1247     Environment ev;
1248     Session session;
1249     ULONG result;
1250     char *cptr, *vrudata, *ptr1;
1251     int     i, j, k;
1252     char cvis[64], dummy[64], dummy1[64];
1253
1254     /* Set defaults in case something goes wrong */
1255     *cvis_tbl = NULL;
1256     *numcvislist = 0;
1257
1258     /* Get the default session for GetFewRecords call */
1259     if (Vesp_get_default_session(object_gbl, &session) != VESP_SUCCESS)
1260     {
1261         log_unparsable ("No default session for loading CVIS table\n");
1262         return;
1263     }
1264
1265     /* Attempt to get the VOX configuration information */
1266     seqseqcp._maximum = 1;
1267     seqseqcp._length = 0;
1268     seqseqcp._buffer = NULL;
1269     result=Vesp_Request_Sync("DS.GetFewRecords", &ev, session, &request, "",
1270             "type=\"srv\" & name=\"VOX\"", &seqseqcp);
1271
1272     if(result == VESP_SUCCESS || result == VESP_PARTIAL_SUCCESS)
1273     {
1274       log_unparsable("Got a DS record for VOX");
1275             /* This is really ugly, but apparently the toolkit
1276              * provides us with no way to parse the configuration
1277              * record for another server.  The "Context" functions
1278              * provided by the toolkit only work for the server's own
1279              * configuration.  The problem is that the GetFewRecords
1280              * call returns a sequence of sequences, one couple in the
1281              * sequences has a name "configuration" and a value
```

Redacted

```
1282              * that is a string representation of yet another
1283              * sequence of couples.  All of this mess unravels that mess.
1284              */
1285          j=0;
1286          *cvis_tbl = (Cvis_List *)vesp_malloc(sizeof(Cvis_List));
1287          for(i=0 ; i < seqseqcp._buffer[0]._length ; i++)
1288          {
1289             if(strcmp(seqseqcp._buffer[0]._buffer[i].name, "configuration") == 0)
1290             {
1291                cptr=seqseqcp._buffer[0]._buffer[i].value;
1292                /* cptr now points at a string which is a representation of
1293                 * a sequence of couples, some of which contain vru names.
1294                 * We'll parse through the string looking at quote marks to
1295                 * delimit names and values.
1296                 */
1297                if((ptr1=strtok(cptr,"\"")) != NULL)
1298                {
1299                while((ptr1=strtok(NULL,"\"")) != NULL)
1300                {
1301                   if(strcmp(ptr1,"vru") == 0)
1302                   {
1303                      /* This a a couple with the name=vru.  The second
1304                       * quote mark starts the value string.
1305                       */
1306                      ptr1=strtok(NULL,"\"");
1307                      vrudata=strtok(NULL,"\"");
1308                      /* vrudata looks like "[Cc]onversant no. name port" */
1309                      if(strncmp(vrudata+1, "onversant ", 10) == 0)
1310                      {
1311                      if(j > 0)
1312                      {
1313                         *cvis_tbl = (Cvis_List *)vesp_realloc(*cvis_tbl,
1314                                  (j+1)*sizeof(Cvis_List));
1315                      }
1316                      sscanf(vrudata,"%s %d %s %s", dummy1, &k, cvis, dummy);
1317                      strcpy((*cvis_tbl)[j].cvis, cvis);
1318                      (*cvis_tbl)[j].handle=NULL;
1319                      j += 1;
1320                      *numcvislist += 1;
1321                      log_unparsable("Added Conversant '%s' to list", cvis);
1322                      }
1323                      /*Or vrudata looks like "[Cc]onversant5 no. name port"*/
1324                      if(strncmp(vrudata+1, "onversant5 ", 11) == 0)
1325                      {
1326                      if(j > 0)
1327                      {
1328                         *cvis_tbl = (Cvis_List *)vesp_realloc(*cvis_tbl,
1329                                  (j+1)*sizeof(Cvis_List));
1330                      }
1331                      sscanf(vrudata,"%s %d %s %s", dummy1, &k, cvis, dummy);
1332                      strcpy((*cvis_tbl)[j].cvis, cvis);
1333                      (*cvis_tbl)[j].handle=NULL;
1334                      j += 1;
1335                      *numcvislist += 1;
1336                      log_unparsable("Added Conversant '%s' to list", cvis);
1337                      }
1338                   } /* If vru record */
1339                } /* Search through "value" string for quote marks */
1340                }
1341             } /* If this is the record named "configuration" */
1342          } /* For all records retrieved from DS */
```

Redacted

```
1343      } /* If DS request is successful */
1344      else
1345      {
1346        log_unparsable("Failed to retrieve DS record for VOX");
1347      }
1348
1349      Vesp_Request_Delete(session, request);
1350
1351      (void)vesp_flush_log();
1352
1353  }
1354
1355  ORBStatus parse_config_parameters (void)
1356  {
1357      Environment env;
1358      Context ctx;
1359      Session session;
1360      SeqCouple *configseq;
1361
1362      /* Get default session, and config from that. */
1363      SCHECK( Vesp_get_default_session(object_gbl, &session));
1364      SCHECK( Session_get_context(session, &env, &ctx));
1365      Context_get_configuration(ctx, &env, &configseq);
1366      if (configseq->_length > 0)
1367      {
1368        /* Nothing to look for currently */
1369        log_unparsable("No special Scripter configuration parameters.");
1370      } /* End if (configseq->_length > 0) */
1371
1372      vesp_couple_seq_delete (configseq);
1373      return VESP_SUCCESS;
1374  }
```

CTIEar.x

```
1   /* Copyright (c) [redacted] AT&T Solutions Customer Care */
2   program CTIEARPROG {
3           version CTIEARVERS {
4                   int TO_D28(string) = 1;
5           }=1;
6   }=99;
```

Redacted

```
 1 /* Copyright (c) [redacted] AT&T Solutions Customer Care */
 2 /*****************************************************************************
 3  * CTIEar - Remote procedure called from the CTI server to pass Conversant
 4  *          connect messages.  Passes whatever it hears on to DIP28.
 5  * J.J. Sager
 6  *****************************************************************************
 7  */
 8 #include <stdio.h>
 9 #include <errno.h>
10 #include <string.h>
11 #include <rpc/rpc.h>
12 #include "CTIEar.h"
13 #include <time.h>
14 #include <sys/timeb.h>
15 #include <sys/ipc.h>
16 #include <sys/sem.h>
17
18 #include "/att/include/mesg.h"
19
20 #define BUFFER_SIZE     200
21 #define     SEM_KEY     284327
22
23 int *to_d28_1(char **msg)
24 {
25       static int ret;
26       int i;
27       int   OutMsgQID;
28       struct      timeb timebuf;
29       char  timestamp[20];
30       char  sysstring[256];
31       key_t sem_key;
32       int   semid;
33
34       struct _OutputMsg {
35             struct mbhdr hd;
36             char buffer[BUFFER_SIZE];
37       } OutMsgBuffer;
38
39       errno = 0;
40       sem_key = SEM_KEY;
41
42       /*************************************************************
43        * Debug mode is turned on by creating a semaphore with ID
44        * equal to SEM_KEY.  Check for it now.
45        *************************************************************
46        */
47       semid = semget(sem_key, 0, IPC_ALLOC);
48       if(semid != -1)
49       {
50          ftime(&timebuf);
51          cftime(timestamp,"%D %T",&timebuf.time);
52          sprintf(sysstring,
53           "/bin/echo 'Received from Scripter %s [%s.%d]' >>/tmp/ctiear.debug",
54           *msg, timestamp, timebuf.millitm);
55          system(sysstring);
56       }
57
58       /*****************************************************
59        * Find the Message Queue for sending data to DIP28
60        *****************************************************
61        */
```

Redacted

```
62        OutMsgQID=msgget(DIP28,0666);
63        if (OutMsgQID < 0)
64        {
65              fprintf(stderr,"Couldn't find message Q ID for DIP28\n");
66              sleep(5);
67              exit(1);
68        }
69
70        /*******************************************************
71         * Set up default values for message header to DIP28
72         *******************************************************
73         */
74        OutMsgBuffer.hd.mchan=0; /* Pretend we're assoc w/ channel 0 */
75        OutMsgBuffer.hd.mtype=1;
76        OutMsgBuffer.hd.morig=100; /* Fake ID  for us */
77        OutMsgBuffer.hd.mcont=100; /* Indicates an "ISDN" record for DIP28 */
78        OutMsgBuffer.hd.mseqno=0;
79
80        strcpy(OutMsgBuffer.buffer, *msg);
81
82
83        i=msgsnd(OutMsgQID,&OutMsgBuffer,sizeof(OutMsgBuffer),1);
84        if(i < 0)
85        {
86              fprintf(stderr,"Couldn't send to DIP28,");
87              fprintf(stderr," return code from msgsnd: %d,", i);
88              fprintf(stderr," errno: %d\n",errno);
89              fflush(stderr);
90              ret=1;
91              return(&ret);
92        }
93        else
94        {
95              ret=0;
96              return(&ret);
97        }
98  }
```

Redacted

isdn.h

```
 1 /*
 2  * Copyright (c) [redacted] AT&T American Transtech Inc.
 3  */
 4 char *Agent_file = "/u/projects/DATA/isdn/agent.file";
 5
 6 /*
 7  *
 8  *    Transaction codes
 9  *
10  */
11 #define ANI  200        /* ani request                */
12 #define RANI 250        /* ani request return code    */
13 #define ISDN_REC  100         /* raw isdn record            */
14 #define AGT_INIT  125         /* reinitialize agent table   */
15 #define LOG_ON    130         /* start logging traffic      */
16 #define LOG_OFF     135       /* stop logging traffic       */
17 #define DIP_TERM  340         /* script termination message */
18 #define ANI_MON   400         /* request complete ani table */
19 #define ANI_STAT  500         /* request statistics on ani  */
20 /*test
21 #define db_pr       printf          /* test*/
22
23 #define MAXCHAN   48            /* Max channels in system */
24 #define MAXCHAR   72            /* Max chars in terminal imput */
25 #define MAXSIZE   80            /* Max bytes in input */
26
27 #define ALARM_VAL 20            /* Initial time-out value */
28 /*
29  *      Message contents
30  */
31
32 #define CONNECT   0x41
33 #define DISCON    0x42
34 #define ABANDON   0x43
35 #define LINKUP    0x44
36 #define LINKDOWN  0x45
37
38 #define LINEX       0x31            /* line extention number */
39 #define TRKGRP    0x32        /* trunk group number    */
40 #define SIDANI    0x39        /* calling party number  */
41 #define CALNAM    0x3A        /* caller's name   ":"   */
42 /*
43  *      Constants and states
44  */
45
46 #define CIDLE     -1            /* Channel is unoccupied */
47 #define CBUSY     1             /*    "        occupied */
48
49 /*
50  *      Offsets and lengths
51  *         Subtract 1 because of 1-numbering
52  */
53
54 #define CAGTNUM   11            /* Pos'n of agent num in connect message */
55 #define AGTLEN    5             /* Length of agent num in message */
56 #define CSIDANI   17            /* Pos'n of ANI digits */
57 #define ANILEN    10            /* Length ...          */
58 #define TKGLEN    3             /* Length of Trk Grp Num */
59 #define DAGTNUM   2-1           /* Pos'n of agent num in disconnect message */
60 #define AAGTNUM   2-1           /* Pos'n of agent num in abandon message */
61 /* Cox [redacted] [redacted] */
```

Redacted

```
62 #define CVDN      34          /* Pos'n of vdn in connect message */
63 #define VDNLEN    5           /* Length of vdn num in connect message */
64 #define CCAL      28          /* Pos'n of called principal extention  */
65 #define CALLEN    5           /* Length of called ext in connect msg  */
66
67 /*********************************************
68 Definition of structures for each host channel
69 *********************************************/
70
71 struct ichan {
72         struct {
73             char status;                /* = -1:unocc; = 0:occupied */
74           char sidagt[AGTLEN+1];    /* 5-digit agent + null term     */
75           char sidcal[CALLEN+1];    /* 5-digit called ext + null term*/
76           char sidvdn[VDNLEN+1];    /* 5-digit VDN + null terminator */
77             char sidani[ANILEN+1];      /* 10-digit ANI num or 3-dig tkg */
78         } call[MAXCHAN];
79 };
80
81 struct agent {
82         char tkgnum[TKGLEN+1];
83         struct {
84             char chan;                  /* Channel, 0-xx */
85             char agtnum[AGTLEN+1];      /* n-digit agent number */
86         } recrd[MAXCHAN];
87 };
88
89
```

Redacted

```
 1 /* Copyright (c) [redacted] AT&T American Transtech (Bob Cox - original code) */
 2 /* Copyright (c) [redacted] AT&T Solutions Customer Care (J. J. Sager - Modify
 3  *                                  for VESP Intfc)
 4  */
 5 /*
 6  * This is the master ISDN DIP.  It receives ISDN data that
 7  * has been received from the CTI/ISDN gateway.  It determines if the message
 8  * applys to this machine by matching the extension in the message
 9  * to a list of extensions in /u/projects/isdn/agent.file.  If the message
10  * applies to this machine, it stores the DNIS and ANI data in a matrix
11  * for the channel that corresponds to that extension.  If an application
12  * uses get_ani.t to retrieve that ISDN data, this DIP returns the data.
13  * A DIPTERM message causes this DIP to put all 0's in the matrix for that
14  * channel.  Also, a Disconnect message from the gateway causes all 0's to
15  * be put there.
16  */
17 #include <stdio.h>
18 #include <sys/types.h>
19 #include <sys/ipc.h>
20 #include <sys/msg.h>
21 #include <signal.h>
22 #include <ctype.h>
23 #include <errno.h>
24 #include <stdio.h>
25 #include "etmsgs/appl_et.h"
26 #include "mesg.h"
27 #include "isdn.h" /* application control of DIP, msgtodip & msgtotsm */
28 #include "hlink.h"
29 #include "hi.h"
30
31 char  tmpagnt[10],tmpani[15];    /* Temp string variables */
32 char  ac_ani[15],tani[15],zani[15]; /* Temp string variables */
33 char  *tptr;                    /* Temporary pointer */
34 struct ichan *shrad, schan;                    /* Channel table pointer */
35 struct agent *shradl, sagent;                   /* Agent table pointer */
36 char  *Logfile = "/u/projects/DATA/isdn/ISDN.log";
37 int   term_sw[MAXCHAN];
38 char  ibuf[1024];
39 char  obuf[2048];
40 long  szmbhdr, szuniv;
41 long  szrmsg;
42 struct      mbhdr *pihdr, *pohdr;
43 char  *pidata, *podata;
44 int   Dipinst;
45 int   no_send = 0;
46 int   log_sw  = 0;
47 long  time();
48 long  ani_cnt = 0;
49 long  noani_cnt = 0;
50 long  nomsg_cnt = 0;
51 long  msg_cnt = 0;
52 int   nomsg_today_cnt = 0;
53 long  start_time;
54 long  this_time;
55 int   rpt_nomsg_cnt = 0;
56
57 main(argc, argv)
58 int argc; char **argv;
59 {
60       int i,lgh;
61       int ret;
```

Redacted

```
62        int chan, orig;
63        short cont;
64        int tchan;        /* channel for not tsm (!TSM)      */
65        shrad = &schan;         /* assign pointers to local memory */
66        shradl = &sagent;       /* assign pointers to local memory */
67
68        for(i=0;i<MAXCHAN;i++)
69              term_sw[i] = 0;
70
71        init( argc, argv );
72
73        agent_init();
74
75          /* Initialize to unavailable and not attached */
76          for (i=0; i < MAXCHAN; i++)
77              clear_chan(i);
78
79        /* Remove any old gateway error files */
80        unlink("/tmp/A.isdnerrs");
81        unlink("/tmp/B.isdnerrs");
82        unlink("/tmp/C.isdnerrs");
83
84        time(&start_time); /* get start time for statistics */
85
86        db_pr("DIP28 Waiting For Messages \n");
87        while(1)
88        {
89                ret = mesgrcv(DIP28, ibuf, 1024, 0,0);
90              if (ret < 0 )
91              {
92                    db_pr("DIP28: MESGRCV FAILURE RET = %d, errno=%d\n",ret,
93                          errno);
94                    continue;
95              }
96              chan = pihdr->mchan;
97              orig = pihdr->morig;
98              cont = pihdr->mcont;
99              pohdr->mchan = pihdr->mchan;
100             /* db_pr("DIP28,rcvd: orig=%2d, ch=%2d, cont=%2d\n",orig,chan,
101                   cont); */
102             if(orig != TSM)
103             {
104                   tchan = chan;
105                   if(tchan < 0)
106                         tchan *= -1;
107             }
108             /*********************************/
109             /** Process the transaction type **/
110             /*********************************/
111             switch(cont)
112             {
113                   case ANI:
114                         pohdr->mcont = RANI;
115                         if(orig == TSM)
116                               i=chan;
117                         else
118                               i = atoi(pidata); /* from anidiptest */
119                         ret = get_ani(podata, i);
120                         lgh = strlen(podata);
121                         term_sw[i] = 0;
122                         szrmsg  = szmbhdr + lgh + 1;
```

Redacted

```
123                     break;
124               case ANI_MON:
125                     pohdr->mcont = RANI;
126                     ret = ani_mon(podata);
127                     lgh = strlen(podata);
128                     szrmsg  = szmbhdr + lgh + 1;
129                     break;
130               case ANI_STAT:
131                     pohdr->mcont = RANI;
132                     ret = ani_stat(podata);
133                     lgh = strlen(podata);
134                     szrmsg  = szmbhdr + lgh + 1;
135                     break;
136               case ISDN_REC:
137                     if(log_sw)
138                     {
139                           /* db_pr("DIP28: LOGGING ISDN RECORD\n"); */
140                           logit(pidata);
141                     }
142                     raw_isdn(pidata);
143                     no_send = 1;
144                     break;
145               case DIP_TERM:
146                     time(&this_time);
147                     db_pr("DIP28: DIP_TERM for Chan: %d [%ld]\n",
148                           chan,this_time);
149                     if(orig == TSM)
150                           i=chan;
151                     else
152                           i = atoi(pidata);
153                     if(!term_sw[i])
154                           clear_chan(i);
155                     no_send = 1;
156                     break;
157               case AGT_INIT:
158                     db_pr("DIP28: AT AGT_INIT\n");
159                     agent_init();
160                     sprintf(podata,"Agent Table Processed\n");
161                     lgh = strlen(podata);
162                     szrmsg  = szmbhdr + lgh + 1;
163                     break;
164               case LOG_ON:
165                     db_pr("DIP28: AT LOG_ON \n");
166                     log_sw = 1;
167                     sprintf(podata,"Logging Started\n");
168                     lgh = strlen(podata);
169                     szrmsg  = szmbhdr + lgh + 1;
170                     break;
171               case LOG_OFF:
172                     db_pr("DIP28: AT LOG_OFF \n");
173                     log_sw = 0;
174                     sprintf(podata,"Logging Stopped\n");
175                     lgh = strlen(podata);
176                     szrmsg  = szmbhdr + lgh + 1;
177                     break;
178               default:
179                     db_pr("DIP28: Invalid Transaction Code (%d)\n",cont);
180                     no_send = 1;
181                     break;
182         }
183         if(no_send)
```

Redacted

```
184             {     /* no reply necessary */
185                   no_send=0;
186                   continue;
187             }
188             if(orig != TSM)
189             {
190                   ret = msgsnd(tchan,obuf,szrmsg,1);
191                   if(ret < 0)
192                         db_pr ("DIP28: !TSM send fail, ret=%d,
errno=%d\n",ret,errno);
193                   time(&this_time);
194                   db_pr ("DIP28: send back(CHAN%d)(%s) [%ld]\n",tchan,
195                         podata,this_time);
196             }
197             else
198             if ((ret = mesgsnd(orig, obuf, szrmsg, 1)) < 0)
199             {
200                   db_pr("DIP28: send fail, ret=%d, errno=%d\n",ret,errno);
201             }
202             else
203             {
204                   /* db_pr("DIP28: SEND TO or=%2d, ch=%2d, cont=%d\n",
205                         orig, chan , pohdr->mcont); */
206                   time(&this_time);
207                   db_pr("DIP28: send to chan:%d (%s) [%ld]\n",chan,podata,
208                         this_time);
209             }
210       } /* END WHILE(1) */
211 } /* END MAIN */
212
213 init( argc, argv )
214 int argc;
215 char **argv;
216 {
217       int ret;
218
219        if (argc < 2) {
220             exit(-1);
221       }
222       Dipinst = atoi( argv[1] );
223       startup( DIP28, Dipinst);
224       while(ret = (mesgrcv( DIP28, ibuf, 0, IPC_NOWAIT )) > 0);
225       szmbhdr = sizeof(struct mbhdr);
226       szuniv = sizeof(struct ms_univ);
227       pihdr = (struct mbhdr *)(ibuf);
228       pohdr = (struct mbhdr *)(obuf);
229       pohdr->mtype = 1;
230       pohdr->morig = DIP28;
231       pidata = ibuf + szmbhdr;
232       podata = obuf + szmbhdr;
233
234 }
235
236 /*****************************************************/
237 /** Process the raw isdn record - taken from DIP15  **/
238 /*****************************************************/
239 raw_isdn(nbuf)
240 char *nbuf;
241 {
242       int i,j;
243
```

Redacted

```
244
245       switch (nbuf[0])
246       {
247             case CONNECT:                    /* Connect message */
248                   msg_cnt++;
249                   strncpy (tmpani,&nbuf[CSIDANI],ANILEN);
250                   tmpani[ANILEN] = 0;
251                   /* Check for no ANI */
252                   strncpy (ac_ani,&nbuf[CSIDANI],ANILEN - 7);
253                   for (i= 0; i<ANILEN; i++)
254                         tani[i] = '9';
255                   tani[i] = '\0';
256                   for (i=0; i<ANILEN - 7; i++)
257                         zani[i] = ' ';
258                   zani[i] = '\0';
259                   if (strcmp(ac_ani,zani) == 0)  /* See if AC is blank */
260                   {
261                         /* no ani, fill with 9's */
262                         strcpy(tmpani,tani);       /* Copy 9's to ANI */
263                         noani_cnt++;
264                   }
265                   else
266                   {
267                         /* we got an ani */
268                         ani_cnt++;
269                   }
270                         j = AGTLEN;              /* No leading spaces */
271                         tptr = &nbuf[CAGTNUM];  /* Start of agent num */
272                         while (isspace(*tptr)) {
273                         j--;                /* Decrement length */
274                         tptr++;                  /* Increment ptr */
275                         }
276                         strncpy (tmpagnt,tptr,j);
277                         tmpagnt[j] = 0;
278
279                         for (i=0; i< MAXCHAN; i++) {
280                               if(strcmp(tmpagnt,shradl->recrd[i].agtnum) == 0)
281                               break;
282                         }
283                         if (i == MAXCHAN)
284                   {
285                         time(&this_time);
286                         db_pr("DIP28: CONNECT: ANI: %s, AGENT: %s, IGNORE
[%ld]\n",tmpani,tmpagnt,this_time);
287                         }
288                   else
289                   {
290                         time(&this_time);
291                         db_pr("DIP28: CONNECT: ANI: %s, AGENT: %s, VDN: %.5s,
CALLED_EXT: %.5s, PORT: %d [%ld]\n",tmpani,tmpagnt,&nbuf[CVDN],&nbuf[CCAL],shradl-
>recrd[i].chan,this_time);
292
293                               j = shradl->recrd[i].chan;
294                               strcpy (shrad->call[j].sidani,tmpani);
295                               shrad->call[j].status=CBUSY;
296                         strncpy(shrad->call[j].sidvdn,&nbuf[CVDN],VDNLEN);
297                         shrad->call[j].sidvdn[VDNLEN] = 0;
298                         strncpy(shrad->call[j].sidagt,&nbuf[CAGTNUM],AGTLEN);
299                         shrad->call[j].sidagt[AGTLEN] = 0;
300                         strncpy(shrad->call[j].sidcal,&nbuf[CCAL],CALLEN);
301                         shrad->call[j].sidagt[CALLEN] = 0;
```

Redacted

```
302                         term_sw[j] = 1;
303
304                         if(nomsg_today_cnt > 0)
305                         {
306                               report_msg(j);
307                               nomsg_today_cnt = 0;
308                         }
309                         }
310                         break;
311                   case DISCON:                    /* Disconnect */
312                     msg_cnt++;
313                         strncpy (tmpagnt,&nbuf[DAGTNUM],AGTLEN);
314                         tmpagnt[AGTLEN] = 0;
315                         j = AGTLEN;               /* No leading spaces */
316                         tptr = &nbuf[DAGTNUM];  /* Start of agent num */
317                         while (isspace(*tptr)) {
318                         j--;                  /* Decrement length */
319                         tptr++;                   /* Increment ptr */
320                         }
321                         strncpy (tmpagnt,tptr,j);
322                         tmpagnt[j] = 0;
323                         for (i=0; i< MAXCHAN; i++) {
324                               if (strcmp(tmpagnt,shradl->recrd[i].agtnum) == 0)
325                               break;
326                         }
327                         if (i == MAXCHAN)
328                     {
329                         time(&this_time);
330                         db_pr("DIP28: DISCONNECT: AGENT: %s, IGNORE
[%ld]\n",tmpagnt,this_time);
331                         }
332                     else
333                     {
334                         time(&this_time);
335                         db_pr("DIP28: DISCONNECT: AGENT: %s,  PORT: %d
[%ld]\n",tmpagnt,shradl->recrd[i].chan,this_time);
336
337                               j = shradl->recrd[i].chan;
338                         clear_chan(j);
339                         term_sw[j] = 0;
340                         if(nomsg_today_cnt > 0)
341                         {
342                               report_msg(j);
343                               nomsg_today_cnt = 0;
344                         }
345                         }
346                         break;
347                   case ABANDON:                   /* Caller Abandon */
348                     msg_cnt++;
349                         strncpy (tmpagnt,&nbuf[CAGTNUM],AGTLEN);
350                         tmpagnt[AGTLEN] = 0;
351                         j = AGTLEN;               /* No leading spaces */
352                         tptr = &nbuf[AAGTNUM];  /* Start of agent num */
353                         while (isspace(*tptr)) {
354                         j--;                  /* Decrement length */
355                         tptr++;                   /* Increment ptr */
356                         }
357                         strncpy (tmpagnt,tptr,j);
358                         tmpagnt[j] = 0;
359                         for (i=0; i< MAXCHAN; i++) {
360                               if (strcmp(tmpagnt,shradl->recrd[i].agtnum) == 0)
```

Redacted

```
361                                 break;
362                           }
363                           if (i == MAXCHAN)
364                     {
365                           time(&this_time);
366                           db_pr("DIP28: ABANDON: AGENT: %s, IGNORE
[%ld]\n",tmpagnt,this_time);
367                           }
368                     else
369                     {
370                           time(&this_time);
371                           db_pr("DIP28: ABANDON: AGENT: %s, PORT: %d
[%ld]\n",tmpagnt,shradl->recrd[i].chan,this_time);
372
373                                 j = shradl->recrd[i].chan;
374                           clear_chan(j);
375                           term_sw[j] = 0;
376                           if(nomsg_today_cnt > 0)
377                           {
378                                 report_msg(j);
379                                 nomsg_today_cnt = 0;
380                           }
381                           }
382                           break;
383                 case LINKUP:                        /* Link up */
384                   msg_cnt++;
385                   db_pr ("DIP28: LINK UP \n");
386                           break;
387                 case LINKDOWN:                      /* Link down */
388                   msg_cnt++;
389                   db_pr ("DIP28: LINK DOWN \n");
390                           break;
391                 default:
392                   db_pr ("DIP28: Invalid ISDN message content: [%c]\n",nbuf[0]);
393             }
394       return(0);
395 }
396
397 clear_chan(channel)
398 int channel;
399 {
400       int j;
401       j = channel;
402       strcpy (shrad->call[j].sidani,"0000000000");
403             strcpy (shrad->call[j].sidvdn,"00000");
404             strcpy (shrad->call[j].sidcal,"00000");
405             strcpy (shrad->call[j].sidagt,"00000");
406             shrad->call[j].status=CIDLE;
407       return(0);
408 }
409
410 /*
411  * this function is to get ANI from memory
412  */
413 get_ani(pfrdip, chanid)
414 char *pfrdip;
415 int chanid;
416 {
417       /*  loads ani info to struct going back to script */
418
419       strcpy(pfrdip, shrad->call[chanid].sidani);     /* callers ani */
```

Redacted

```
420       strcat(pfrdip, shrad->call[chanid].sidagt);     /* agent ext */
421       strcat(pfrdip, shrad->call[chanid].sidcal);     /* called ext */
422       strcat(pfrdip, shrad->call[chanid].sidvdn);     /* vdn */
423
424             if(shrad->call[chanid].status == CIDLE)
425       {
426             nomsg_cnt += 1;
427             nomsg_today_cnt += 1;
428             time(&this_time);
429             db_pr("DIP28: ***CALL WITH NO GATEWAY MESSAGE*** chan: %d
[%ld]\n",chanid,this_time);
430
431             /* If we get more than 10 nomsg's in a row, log */
432             /* a higher priority message */
433             if(nomsg_today_cnt > 10)
434             {
435                   db_pr("DIP28: **EXCESSIVE CALLS WITH NO GATEWAY MESSAGES**\n");
436                   report_nomsg(chanid);
437             }
438       }
439
440       return(0);
441 }
442
443 /*  loads ani info to struct going back to script */
444 ani_mon(monout)
445 char *monout;
446 {
447       int i;
448       int pos;
449       pos=0;
450       for(i=0;i<MAXCHAN;i++)
451             {
452             sprintf(&monout[pos],"(%02d):%s:%s|",
453                   i,
454                   shrad->call[i].sidani,  /* callers ani */
455                   shrad->call[i].sidvdn); /* called vdn */
456             pos=strlen(monout);
457             }
458
459       return(0);
460 }
461
462 /*  Process ani statistics */
463 ani_stat(statout)
464 char *statout;
465 {
466       long diff;
467       long minutes;
468       long hours;
469       long days;
470       char timebuf[30];
471
472       time(&this_time);
473       diff = this_time - start_time;
474       minutes = diff / 60;
475       hours = minutes / 60;
476       days = hours / 24;
477       cftime(timebuf,"%B %d, %Y %H:%M:%S",&start_time);
478
```

Redacted

```
  479       sprintf(statout,"Ani=%d, No Ani=%d, Mess=%d, No Mess=%d, Since %s
(%d:%02d:%02d:%02d)",ani_cnt,noani_cnt,msg_cnt,nomsg_cnt,timebuf,days,(hours%24),(minutes%
60),(diff%60));
  480       return(0);
  481 }
  482
  483 /*
  484  * This process creates and initializes both areas of shared memory.
  485  *    It then initializes all entries in the channel status table to un-
  486  *    occupied.  Finally, it reads disk file /vs/data/agent.file and
  487  *    creates the table which maps agent extension number to incoming
  488  *    channel.
  489  */
  490 agent_init()
  491 {
  492       FILE *fopen(), *fptr;
  493         int  i,j;
  494
  495         char inagent[10],inchan[5];             /* temp storage */
  496         char intkgnum[10];                      /* temp storage */
  497
  498         char tmpchan;
  499
  500
  501         /* Read disk file and establish host links */
  502
  503         fptr = fopen (Agent_file,"r");
  504       db_pr ("DIP28: Opening agent file '%s'\n",Agent_file);
  505         if (fptr == NULL) {
  506       db_pr("DIP28: NO AGENT FILE (%s)\n",Agent_file);
  507       sleep(5);
  508             exit (-1);
  509         }
  510         if ((j = fscanf (fptr, "%s", intkgnum)) != 1) {
  511                   db_pr("DIP28: Missing trunk group number, tokens = %d\n",j);
  512             fclose(fptr);
  513             exit(3);
  514         }
  515       /* db_pr ("Trunk group number = [%s]\n\n",intkgnum); */
  516         strcpy (shradl->tkgnum, intkgnum);      /* Copy to shr mem */
  517         for (i=0; i < MAXCHAN; i++) {
  518             if ((j = fscanf (fptr, "%s %s", inagent, inchan)) != 2) {
  519                 if (j == -1) {
  520                     db_pr ("DIP28: Agent File Initialization complete!\n");
  521                     return (0);
  522                 }
  523                 db_pr("DIP28: Incorrect agent/chan table, tokens = %d\n",j);
  524             fclose(fptr);
  525                 exit(3);
  526             }
  527             if (((tmpchan = atoi (inchan)) > MAXCHAN)) {
  528                 db_pr("DIP28: Incorrect agent/chan table, tokens = %d\n",j);
  529             fflush (stdout);
  530                 continue;
  531             }
  532       db_pr ("\tExt = [%s], chan = %d\n", inagent, tmpchan);
  533             strcpy(shradl->recrd[i].agtnum, inagent);   /* Copy to shr mem */
  534             shradl->recrd[i].chan = tmpchan;
  535         }
  536
  537         fclose (fptr);
```

Redacted

```
538         return (0);
539 }
540
541 logit(rec)
542 char *rec;
543 {
544       int lgh,fd;
545       lgh = strlen(rec);
546       if(lgh > 80)
547             lgh = 80;
548       fd = open(Logfile,2);
549       if((fd < 0) && (errno == 2))
550             fd = creat(Logfile,0666);
551       if(fd < 0)
552             fd = open("/dev/console",1);
553       lseek(fd,0,0);
554       lseek(fd,0,2);
555       write(fd,rec,lgh);
556       close(fd);
557       return(0);
558 }
559
560 report_nomsg(chan)
561 int   chan;
562 {
563       char  command[50];
564
565       /* Call the isdn_error program to report a gateway problem */
566       if(rpt_nomsg_cnt == 0)
567       {
568             sprintf(command,"/etc/etc/isdn_error %d nomsg",chan);
569             db_pr("DIP28: Calling '%s'\n",command);
570             system(command);
571       }
572
573       /* Don't do this too often */
574       rpt_nomsg_cnt++;
575
576       if(rpt_nomsg_cnt > 10)
577             rpt_nomsg_cnt=0;
578 }
579
580 report_msg(chan)
581 int   chan;
582 {
583       char  command[50];
584
585       /* Call the isdn_error program to report a good call */
586       sprintf(command,"/etc/etc/isdn_error %d msg",chan);
587       db_pr("DIP28: Calling '%s'\n",command);
588       system(command);
589       rpt_nomsg_cnt=0;
590 }
```

Redacted

route_call Script Builder code

```
     start:

     ###################################################
     #_ ROUTE CALL APPLICATION
     #_
     #_ PROGRAMMER:  JEFF CREWS
     #_ DATE:        [redacted]
     #_
     #_ PURPOSE:  THE PURPOSE OF THIS APPLICATION IS TO
     #_   TAKE AN INCOMING CALL AND PERFORM A NEXTSCRIPT
     #_   TO THE APPLICATION THE CALLER WANTED BASED ON
     #_   THE 800 NUMBER THE CALLER DIALED.
     ###################################################

     ###################################################
     #_ Modified [redacted] by J.J. Sager for addition
     #_   of Environment specification table (ROUTE_ENV)
     #_   and VESP enhancements.
     ###################################################

     #_ Trap Caller Aborts to track VESP interaction
1.   External Function
        Function Name: callabort

     #_ Initialize Variables
2.   Set Field Value
           Field: CALLERS_INCOMING_NUMBER = "0000000000"
           Field: ATTEMPTED_DNIS_NUMBER = "00000"
           Field: ORIGINATING_NUMBER = "000000000"

     ###################################################
     VESP_INITIATE:
     ###################################################

3.   Read Table
        Table Name:  ROUTE_ENV     Search From Beginning
           Field: RT_ENV_PARAM = "vesp_enabled"
4.   Evaluate
     If RT_ENV_VALUE   != "yes"
5.      Goto GET_DNIS_NUMBER
     End Evaluate

6.   Set Field Value
           Field: NEWCALL_NEEDED = "yes"
7.   Answer Phone
8.   External Function
        Function Name: vesp_newcall
        Use Arguments: VDUID  "route_callA"  $UNIX_TIME  $CHANNEL_NUMBER
        Return Field:  RTN_CODE
9.   Evaluate
     If RTN_CODE   = 0
10.     Set Field Value
              Field: NEWCALL_SENT = "yes"
11.     External Function
           Function Name: putvduid
           Use Arguments: VDUID
           Return Field:  RTN_CODE
     End Evaluate

     ###################################################
     GET_DNIS_NUMBER:
```

Redacted

route_call Script Builder code

```
     ##################################################

12.  External Function
        Function Name: get_ani
        Use Arguments: ISDN_RECORD
        Return Field:  RTN_CODE
13.  External Function
        Function Name: substring
        Use Arguments: DNIS_NUMBER  ISDN_RECORD  20  5
        Return Field:  RTN_CODE
14.  External Function
        Function Name: substring
        Use Arguments: ANI_NUMBER  ISDN_RECORD  1  10
        Return Field:  RTN_CODE
15.  Evaluate
     If DNIS_NUMBER   = "00000"
16.     External Function
           Function Name: sleep
           Use Arguments: 5
           Return Field:  RTN_CODE
17.     External Function
           Function Name: get_ani
           Use Arguments: ISDN_RECORD
           Return Field:  RTN_CODE
18.     External Function
           Function Name: substring
           Use Arguments: DNIS_NUMBER  ISDN_RECORD  20  5
           Return Field:  RTN_CODE
19.     External Function
           Function Name: substring
           Use Arguments: ANI_NUMBER  ISDN_RECORD  1  10
           Return Field:  RTN_CODE
20.     Evaluate
     If DNIS_NUMBER   = "00000"
21.        Goto NEED_INCOMING_NUMBER
        End Evaluate
     End Evaluate
     #_ Handle Switch A, B, or C VDNs (4 or 5 digits)
22.  External Function
        Function Name: substring
        Use Arguments: BLANK_CHECK  DNIS_NUMBER  0  1
        Return Field:  RTN_CODE
23.  Evaluate
     If BLANK_CHECK   = " "
24.     External Function
           Function Name: parse
           Use Arguments: BLANK_CHECK  DNIS_NUMBER  " "
           Return Field:  RTN_CODE
     End Evaluate
25.  Read Table
        Table Name:  ROUTE_APPL     Search From Beginning
           Field: route_appl_dnis_number = DNIS_NUMBER
26.  Evaluate
     If $MATCH_FOUND   > 0
27.     Set Field Value
              Field: HOLD_DNIS_NUMBER = route_appl_dnis_number
              Field: HOLD_INCOMING_NUMBER = route_app_800_number
              Field: HOLD_APPLICATION_NAME = route_appl_name
              Field: HOLD_DO_CLEAR_ANI = route_appl_do_clear_ani
28.     Goto CALL_NEXT_SCRIPT
     Else
```

Redacted

route_call Script Builder code

```
29.      Goto NEED_INCOMING_NUMBER
      End Evaluate

      ##################################################
      NEED_INCOMING_NUMBER:
      ##################################################

      #_ DNIS NUMBER NOT FOUND
30.   Set Field Value
            Field: ATTEMPTED_DNIS_NUMBER = ANI_NUMBER
            Field: ORIGINATING_NUMBER = ANI_NUMBER
            Field: NUMBER_NOT_FOUND = NUMBER_NOT_FOUND + 1
31.   Answer Phone
32.   Announce
         Speak Without Interrupt
            Phrase: "we are having trouble completing your call"
33.   Set Field Value
            Field: ONE = 1
            Field: ONE_EIGHT_HUNDRED = 1800
            Field: ONE_EIGHT_EIGHT_EIGHT = 1888
      #_ Get incoming number
34.   Prompt & Collect
         Prompt
            Speak With Interrupt
               Phrase: "please re-enter the number"
         Input
            Caller Input Field: CALLERS_INCOMING_NUMBER
            Max Number Of Digits: 11
         Checklist
            Case: "nnnr"
               Continue
            Case: "Not On List"
               Speak With Interrupt
                  Phrase: "invalid entry"
               Reprompt
            Case: "Initial Timeout"
               Speak With Interrupt
                  Phrase: "too few digits"
               Reprompt
            Case: "Too Few Digits"
               Speak With Interrupt
                  Phrase: "too few digits"
               Reprompt
            Case: "No More Tries"
               Speak With Interrupt
                  Phrase: "unable to complete call, please try again"
               Quit
      End Prompt & Collect
      #_ Write ERROR record
35.   Modify Table
         Table Name: ROUTE_ERROR Operation: Add
            Field: route_attempted_800 = CALLERS_INCOMING_NUMBER
            Field: route_attemted_dnis = ATTEMPTED_DNIS_NUMBER
            Field: route_originating_num = ORIGINATING_NUMBER
36.   Read Table
         Table Name:  ROUTE_APPL     Search From Beginning
            Field: route_app_800_number = CALLERS_INCOMING_NUMBER
37.   Evaluate
      If $MATCH_FOUND   > 0
38.      Set Field Value
               Field: HOLD_DNIS_NUMBER = route_appl_dnis_number
```

Redacted

route_call Script Builder code

```
                Field: HOLD_INCOMING_NUMBER = route_app_800_number
                Field: HOLD_APPLICATION_NAME = route_appl_name
                Field: HOLD_DO_CLEAR_ANI = route_appl_do_clear_ani
          #_ Check for Multiple Incoming Numbers in table
39.       Read Table
             Table Name:  ROUTE_APPL
                Field: route_app_800_number = CALLERS_INCOMING_NUMBER
40.       Evaluate
       If $MATCH_FOUND   > 0
41.          Set Field Value
                   Field: NOT_SURE_OF_DNIS_FLAG = "Y"
          End Evaluate
42.       Goto FIX_ISDN_RECORD
       End Evaluate
       #_ Check for an abbreviated incoming 800 number
43.    Evaluate
       If $CI_NO_DIGS_GOT   = "7"
          #_ Add 1800 in front of 7 digit incoming number
44.       External Function
             Function Name: concat
             Use Arguments: TRY_NEW_CALLERS_INC_NUM  ONE_EIGHT_HUNDRED
CALLERS_INCOMING_NUMBER  0
             Return Field:  RTN_CODE
45.       Read Table
             Table Name:  ROUTE_APPL     Search From Beginning
                Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
46.       Evaluate
       If $MATCH_FOUND   > 0
47.          Set Field Value
                   Field: HOLD_DNIS_NUMBER = route_appl_dnis_number
                   Field: HOLD_INCOMING_NUMBER = route_app_800_number
                   Field: HOLD_APPLICATION_NAME = route_appl_name
                   Field: HOLD_DO_CLEAR_ANI = route_appl_do_clear_ani
             #_ Check for Multiple Incoming Numbers in table
48.          Read Table
                Table Name:  ROUTE_APPL
                   Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
49.          Evaluate
       If $MATCH_FOUND   > 0
50.             Set Field Value
                      Field: NOT_SURE_OF_DNIS_FLAG = "Y"
             End Evaluate
51.          Goto FIX_ISDN_RECORD
          End Evaluate
       End Evaluate
       #_ Check for an abbreviated incoming 888 number
52.    Evaluate
       If $CI_NO_DIGS_GOT   = "7"
          #_ Add 1888 in front of 7 digit incoming number
53.       External Function
             Function Name: concat
             Use Arguments: TRY_NEW_CALLERS_INC_NUM  ONE_EIGHT_EIGHT_EIGHT
CALLERS_INCOMING_NUMBER  0
             Return Field:  RTN_CODE
54.       Read Table
             Table Name:  ROUTE_APPL     Search From Beginning
                Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
55.       Evaluate
       If $MATCH_FOUND   > 0
56.          Set Field Value
                   Field: HOLD_DNIS_NUMBER = route_appl_dnis_number
```

Redacted

route_call Script Builder code

```
                Field: HOLD_INCOMING_NUMBER = route_app_800_number
                Field: HOLD_APPLICATION_NAME = route_appl_name
                Field: HOLD_DO_CLEAR_ANI = route_appl_do_clear_ani
          #_ Check for Multiple Incoming Numbers in table
57.       Read Table
             Table Name:  ROUTE_APPL
                Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
58.       Evaluate
     If $MATCH_FOUND   > 0
59.          Set Field Value
                   Field: NOT_SURE_OF_DNIS_FLAG = "Y"
          End Evaluate
60.       Goto FIX_ISDN_RECORD
       End Evaluate
     End Evaluate
     #_ Check for an abbreviated incoming number (No 1)
61.  Evaluate
     If $CI_NO_DIGS_GOT   = "10"
        #_ Add 1 in front of 10 digit incoming number
62.     External Function
          Function Name: concat
          Use Arguments: TRY_NEW_CALLERS_INC_NUM  ONE  CALLERS_INCOMING_NUMBER  0
          Return Field:  RTN_CODE
63.     Read Table
          Table Name:  ROUTE_APPL     Search From Beginning
             Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
64.     Evaluate
     If $MATCH_FOUND   > 0
65.       Set Field Value
                Field: HOLD_DNIS_NUMBER = route_appl_dnis_number
                Field: HOLD_INCOMING_NUMBER = route_app_800_number
                Field: HOLD_APPLICATION_NAME = route_appl_name
                Field: HOLD_DO_CLEAR_ANI = route_appl_do_clear_ani
          #_ Check for Multiple Incoming Numbers in table
66.       Read Table
             Table Name:  ROUTE_APPL
                Field: route_app_800_number = TRY_NEW_CALLERS_INC_NUM
67.       Evaluate
     If $MATCH_FOUND   > 0
68.          Set Field Value
                   Field: NOT_SURE_OF_DNIS_FLAG = "Y"
          End Evaluate
69.       Goto FIX_ISDN_RECORD
       End Evaluate
     End Evaluate
     #_ Didn't find an application to call
70.  Announce
        Speak With Interrupt
          Field: CALLERS_INCOMING_NUMBER As Cmmf
          Phrase: "is not on our system"
          Phrase: "unable to complete call, please try again"
71.  Goto TERMINATE_CALL

     ###################################################
     FIX_ISDN_RECORD:
     ###################################################

72.  External Function
        Function Name: substring
        Use Arguments: KEEP_FIRST15_ISDN_RECORD  ISDN_RECORD  0  15
        Return Field:  RTN_CODE
```

Redacted

route_call Script Builder code

```
73.  Set Field Value
           Field: REBUILD_ISDN_RECORD = KEEP_FIRST15_ISDN_RECORD
74.  External Function
        Function Name: length
        Use Arguments: HOLD_DNIS_NUMBER
        Return Field:  DNIS_LENGTH
     #_ (DNIS_LENGTH = 4) we account for 4 digit VDNs
75.  Evaluate
     If DNIS_LENGTH    = 4
76.     External Function
           Function Name: concat
           Use Arguments: REBUILD_ISDN_RECORD  REBUILD_ISDN_RECORD  " "  0
           Return Field:  RTN_CODE
     End Evaluate
77.  External Function
        Function Name: concat
        Use Arguments: REBUILD_ISDN_RECORD  REBUILD_ISDN_RECORD  HOLD_DNIS_NUMBER  0
        Return Field:  RTN_CODE
78.  Evaluate
     If DNIS_LENGTH    = 4
79.     External Function
           Function Name: concat
           Use Arguments: REBUILD_ISDN_RECORD  REBUILD_ISDN_RECORD  " "  0
           Return Field:  RTN_CODE
     End Evaluate
80.  External Function
        Function Name: concat
        Use Arguments: REBUILD_ISDN_RECORD  REBUILD_ISDN_RECORD  HOLD_DNIS_NUMBER  0
        Return Field:  RTN_CODE
81.  Set Field Value
           Field: ISDN_RECORD = REBUILD_ISDN_RECORD
82.  Goto CALL_NEXT_SCRIPT

     ###################################################
     CALL_NEXT_SCRIPT:
     ###################################################

     #_ See if HOLD_APPLICATION_NAME is just an alias
83.  Read Table
        Table Name:  ROUTE_ALIAS     Search From Beginning
           Field: route_alias_name = HOLD_APPLICATION_NAME
84.  Evaluate
     If $MATCH_FOUND    > 0
85.     Set Field Value
              Field: HOLD_EXECUTE_NAME = route_alias_execute_pgm
     Else
86.     Set Field Value
              Field: HOLD_EXECUTE_NAME = HOLD_APPLICATION_NAME
     End Evaluate
     #_ Clear the ISDN Record for the next call
     #_     if no subprograms to route call do get_ani
87.  Evaluate
     If HOLD_DO_CLEAR_ANI    = "Y"
88.     External Function
           Function Name: clear_ani
           Return Field:  RTN_CODE
     End Evaluate
     #_ Check flag for multiple incoming number carrying
     #_     different DNISs.
89.  Evaluate
     If NOT_SURE_OF_DNIS_FLAG    = "Y"
```

Redacted

route_call Script Builder code

```
90.      Set Field Value
               Field: HOLD_APPLICATION_NAME = "???"
      End Evaluate
91.   External Action: Execute
         Application_Name:  HOLD_EXECUTE_NAME
         Write_Call_Data_Record:  "yes"
         Argument_1:  ISDN_RECORD
         Argument_2:  HOLD_APPLICATION_NAME
      End External Action
92.   Announce
         Speak With Interrupt
            Phrase: "unable to complete call, please try again"
93.   Goto TERMINATE_CALL

      ####################################################
      TERMINATE_CALL:
      abort_call:
      ####################################################

94.   Evaluate
      If NEWCALL_NEEDED   = "yes"
95.      Evaluate
      If NEWCALL_SENT   != "yes"
96.         External Function
               Function Name: vesp_newcall
               Use Arguments: VDUID  "route_callA"  $UNIX_TIME  $CHANNEL_NUMBER
               Return Field:  RTN_CODE
97.         Evaluate
      If RTN_CODE   = 0
98.            External Function
                  Function Name: putvduid
                  Use Arguments: VDUID
                  Return Field:  RTN_CODE
            End Evaluate
         End Evaluate
      End Evaluate
99.   Disconnect
100.  Quit
```

Redacted

```
 1 /* Copyright (c) [redacted] AT&T American Transtech Inc.
 2  *
 3  *  isdn info from DIP28
 4  *
 5  *  Record=aaaaaaaaaa222223333344444N
 6  *
 7  *  a=ani, could be all 9 or 0 for no-ani
 8  *  2=agent or conversant port
 9  *  3=called  extention
10  *  4=called VDN
11  *  N=following null character
12  *
13  *  Note:  This external function does not clear the ani information.
14  *         clear_ani should be used after a get_ani if no subprograms will
15  *         be using ISDN information.
16  *
17  */
18
19 L__get_ani:
20
21       dbase(28, 200, ch.F__TEMP, 40, ch.F__TEMP, 0)
22       trace(imm.44444,ch.F__TEMP)
23       strcpy(*ch.3, ch.F__TEMP)
24       rts()
```

Redacted

TAB 3: Alternative Solutions

Alternative 1: Utilize Definity CONVERSE Vector Step

Overview

This alternative could be implemented with no CTI servers required. In our normal Definity Vector design, the calls are routed to a Conversant VRU by queuing to an ACD split (each Conversant is defined to a single split). The design of this method involved changing the vectors so that instead of queuing to a split, they could call the Conversant using the CONVERSE step. By design, CONVERSE allows the Vector to pass two pieces of data (ANI and DNIS) via DTMF to the Conversant. The route_call program could be modified to accept the DTMF tones and proceed with looking up the DNIS in a table and executing the correct customer application. In addition, because the CONVERSE vector function keeps the vector in control of the call throughout the entire life of the call, the Conversant customer application program would either have to give up control of the disposition of the call (e.g. it could not directly transfer the caller to an agent, but instead would have to pass that control back to the vector) or the route_call application would pass the receiving extension back to the vector and the vector would transfer the call to that extension directly rather than maintaining control.

Pros

This design required no CTI services outside of those provided by Lucent in the Definity and the Conversant.

Cons

Required a great deal of time to complete. Almost 4 to 5 seconds were wasted on every call passing DTMF and establishing control.
Required that every existing vector be re-written and extensive training for Vector programmers.
Only available for Lucent Definity PBX switches.

Alternative 2: Utilize VESP VDU Information to Gather DNIS and ANI

Overview

The design of this alternative involved using the VESP functions as they are supplied "out-of-the-box" from Nabnasset. The route_call application (which is always assigned to every shared port), upon going off-hook, would issue a "newcall" method call to obtain this call's unique VDU ID, and then two "get_vdu" method calls to obtain the DNIS and ANI for this call. It could then perform a table lookup and start the correct application in the normal manner.

Pros

This design required the least amount of development because it uses off-the-shelf method calls.

Cons

Required a great deal of time to complete a total of three VESP method calls after the phone was already taken off hook (1 to 2 seconds were measured in proto-typing).
Increased Conversant-to-VESP traffic significantly for every call.

Redacted

TAB 4: Original Request for Proposal to Nabnasset Corp.

The following is a request we made to Nabnasset Corp. for a proposal to supply us with a solution to our Port Sharing needs. We were asking for a method within the existing VESP architecture that would allow ANI and DNIS information to be passed to the Conversant unsolicited. This, we believed, would save significant time over the alternative solutions we had already proposed. Ultimately, Nabnasset did not fulfill this request,. Instead, we came up with another solution (different from those documented above) based on face-to-face discussions with Nabnasset programmers regarding the VESP architecture.

Before we arrived at our own solution, Nabnasset referred us to another software vendor (Gold Systems) whom they believed to be developing a similar solution. We requested a description of their solution and found it to be essentially equivalent to our Alternative 2 above.

Issues Regarding the Use of VESP/Conversant Interface for Port Sharing at ATI

Executive Summary

The Conversant systems at ATI utilize a call's ISDN information to dynamically assign applications to channels. We call this "Port Sharing" because any given Conversant port can be used by any client application to provide VRU service. This is much more efficient than hard-assigning client applications to individual ports and allows us to achieve very high port utilization. We would like to utilize VESP to gather the DNIS and ANI information prior to assigning the correct client application to a port, but there are performance obstacles to using VESP for this purpose. We would like to request a change to the VESP/Conversant interface that would make port sharing using VESP services possible.

Current Environment

Currently at ATI all 794 Conversant ports are shared by multiple applications. Based on the called DNIS, the correct application is dynamically assigned to the correct ACD extension that is wired to a Conversant port. This is done without connecting the Conversants to the PBXs with BRI links (because we have too many Conversant systems) and without using the CONVERSE vector step to pass the DNIS using DTMF (because it adds to much time to the call duration). Instead of those methods, we use a CTI solution outside of VESP that involves gateway systems which communicate with the switch using ASAI/BRI links and pass the incoming call information to the Conversants during call setup.

The importance of the words "during call setup" can not be stressed enough. The DNIS and ANI information is collected by the Conversant (via the gateway) before the port even detects ringing. This speeds the call setup process. Since the information is being passed through the gateway in parallel with the PBX sending a ring signal to the Conversant port, virtually no time is added to the total call duration (thus saving toll charges).

The information from the ASAI gateway is passed via TCP/IP sockets and is collected by a DIP on Conversant and stored in temporary memory. When the port detects ringing, a locally developed application called route_call, which is assigned to the port, requests the ASAI information from the DIP and uses a table lookup to start the correct application on that port. On our VESP-enabled ports, route_call also initializes the call with the VOX server by issuing the "newcall" method and obtains the VDU ID for that call. The ASAI information and VDU ID are passed to the intended application program as needed.

Problems Utilizing VESP Environment

An ideal VESP implementation at ATI would eliminate the ASAI gateway on VESP-enabled systems and use only the VESP services for application assignment. The ASAI gateway is redundant since the information it passes to the Conversant is a duplication of the information known by the VESP services. Unfortunately, the design of the VESP interface to Conversant requires that the Conversant initiate all information flow. All VDU data must be specifically requested by Conversant. That is, an event on Conversant generated by the caller (e.g. the ringing of the phone port) drives requests by Conversant of the VESP services. No event outside of the Conversant (e.g. an IncomingCall event detected by Telephony Server) causes any information to flow to the Conversant.

Under the current VESP design, in order for us to eliminate the ASAI gateway from our system, the Conversant would have to wait for the port to detect ringing before initiating requests for DNIS and ANI information from the VDU. The Conversant would have to

Redacted

generate at least three transactions with VESP: 1) the newcall transaction to obtain the VDU ID; 2) a getvdu transaction to obtain the DNIS; 3) another getvdu to obtain the ANI. (Note: the new version 5 VESP DIP that we just received allows for multiple pieces of data to be retrieved with one getvdu transaction, but we have not yet begun to use this feature). These transactions would add some amount of time to the call duration, perhaps a second or more, which when taken on a per call bases may not seem significant, but when multiplied by the 2.5 million Conversant calls we take each month becomes very significant.

Request for a Solution

When viewed in the most fundamental terms, what we are asking is that the Conversant be treated like any other VESP client. The screen-pop function works on an agent workstation because the necessary information is sent to the workstation unsolicited before the agent hears the zip-tone (or ringing). We are asking for a screen-pop-like methodology for Conversant. We don't wish to give up the current interface - it makes since for Conversant to drive most communications with the VESP servers because, by definition, the Conversant reacts to real-time events presented by the callers. We would like an addition to the current interface that somehow allows the VOX server to send information on to the Conversant, unsolicited, at the time of an IncomingCall event as detected by the Telephony server. At a minimum the information content should be DNIS, ANI, and ACD extension. Ideally, the information passed should be user definable. We could envision a use for additional information (e.g. caller's name, account number, account balance, etc.) to be passed before the call begins. The VRU might answer: "Thanks for calling XYZ Corp. Mr. Smith. When you last called you requested an account statement be mailed to you. How can we serve you today?"

Redacted

TAB 5: Milestone Time Line

[redacted] - Initial requirements for Port Sharing method developed and design studies began.

[redacted] - Some design solutions proposed and proto-typing and research of solution validity begins. None of the initial solutions met all requirements.

[redacted] - Request for proposal submitted to Nabnasset.
Nabnasset refers us to Gold Systems.
Gold Systems proposal received and reviewed and determined to not fulfill requirements.

[redacted] - CTI development team travels to Acton, MA and has face-to-face consultation/training meetings with Nabnasset developers. The architecture of the Scripter Server (which is typically used for Caller Qualification) was discussed in detail. The sections of the server which are user-customizable (custom.c) were described and examples were presented. During these discussions, a largely un-documented method within the Scripter custom.c code was discovered (Script.Qualify1).

[redacted] - Upon returning home and doing more research a new design was proposed using the Script.Qualify1 method and RPC fucntionality.
All previous designs and this new design were weighed and the decision was made to proceed with the new design.

[redacted] - Code is written and the concept and functionality are proven.
Testing of superiority over alternative methods is performed.

[redacted] - Solution is tested by separate Q/A team.

[redacted] - Coding and Testing are completed and an implementation plan is proposed and reviewed with Conversant Support team.

[redacted] - Solution is moved to limited production use and performance is tested under load conditions.

[redacted] - Performance is approved and full production implementation roll-out is begun.

Redacted

Qualities That Make Our Port-Sharing Solution Unique

1. **Port Re-use**
   Any VRU port in the call center can be used for any application. Beyond the initial database record that relates a DNIS to an application, there is no special set-up required for each project or each 800 number to re-use a VRU port.
2. **Out-of-Band Communication**
   Our system transfers the call information (i.e. ANI and DNIS) to the VRU system on a communications channel that is completely separate from the voice channel being used by the caller to talk to the VRU. This makes the port-sharing effort very efficient because the call data is communicated in parallel to the call being switched to the VRU and the VRU can know which application to start even before the phone "rings". Out-of-Band communication also simplifies the administration of the call flow – that is, the call flow designer does not have to take port-sharing into consideration when deciding to include a VRU in the design.
3. **Client/Server Architecture**
   The VRU is simply one part of a larger system that encompasses many clients and servers (PBX, CTI Server, Administration Servers, etc.). Adding new VRU's or new CTI servers is routine and does not require any additional programming or infrastructure changes.
4. **Application Program Transparency**
   VRU application "scripts" need not be aware of the port-share system. A script can be designed and tested without consideration being given to the port-sharing. Then, when implemented into production, the script will automatically work under the port-sharing system. If, however, the application wishes to make use of the call data (ANI and DNIS), that data can be procured from the port-sharing system.
5. **Data Cache**
   The call data is communicated once to the VRU when the call is originally set up and is cached there for subsequent use by the port-sharing system or by the application scripts if desired.

Modules of the design that are unique to our invention:

1. **Customizations of the Script.Qualify1 method within the Scripter server.**
2. **Creation of the CTIEar module to communicate, out-of-band, with the CTI server.**
3. **Creation of the dip28_isdn program to collect and store the information passed from the CTI server.**
4. **Creation of the route_call IVR script program to relate the DNIS information to a client application and then start that client application on the IVR port.**

CinLibrary/1324481.3